THE HARPY SPEECH RECOGNITION SYSTEM

CARNEGIE-MELLON UNIVERSITY
PITTSBURGH, PENNSYLVANIA

APRIL 1976

The HARPY Speech Recognition System

Bruce T. Lowerre

April, 1976

DEPARTMENT
of
COMPUTER SCIENCE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR - TR - 77 - 0021	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) THE HARPY SPEECH RECOGNITION SYSTEM		5. TYPE OF REPORT & PERIOD COVERED Interim
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Bruce T. Lowerre		8. CONTRACT OR GRANT NUMBER(s) F44620-73-C-0074
9. PERFORMING ORGANIZATION NAME AND ADDRESS Carnegie-Mellon University Computer Sciece Dept. Pittsburgh, PA 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61101D AO 2466
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd Arlington, VA 22209		12. REPORT DATE April 1976
		13. NUMBER OF PAGES 120
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Air Force Office of Scientific Research (NM) Bolling AFB, DC 20332		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Harpy connected speech recognition system is the result of an attempt to understand the relative importance of various design choices of two earlier speech recognition systems developed at Carnegie-Mellon University: The Hearsay-I system and the Dragon system. Knowledge is represented in the Hearsay-I system as procedures and in the Dragon system as a Markov network with a-priori transition probabilities between states. Hearsay-I uses a best-first search strategy of the

syntactic paths while Dragon searches all the possible syntactic (and acoustic) paths through the network in parallel to determine the globally optimal path. Hearsay-I uses segmentation and labeling to reduce the effective utterance length while Dragon is a segmentation-free system. Systematic performance analysis of various design choices of these two systems resulted in the HARPY system, in which knowledge is represented as a finite state transition network but without the a-priori transition probabilities. Harpy searches only a few "best" syntactic (and acoustic) paths in parallel to determine the optimal path, and uses segmentation to effectively reduce the utterance length, thereby reducing the number of state probability updates that must be done.

Several new heuristics have been added to the HARPY system to improve its performance and speed: detection of common sub-nets and collapsing them to reduce overall network size and complexity, eliminating the need for doing an acoustic match for all phonemic types at every time sample, and semi-automatic techniques for learning the lexical representations (that are needed for a steady-state system of this type) and the phonemic templates from training data, thus automatically accounting for the commonly occurring intra-word coarticulation and juncture phenomena. Inter-word phenomena are handled by the use of juncture rules which are applied at network generation time, thereby eliminating the need for repetitive and time consuming application of phonological rules during the recognition phase. State transition probabilities are calculated dynamically during the recognition phase from speech dependent knowledge rather than a-priori from statistical measurements.

The system was tested on four sets of data containing 240 sentences (80 for training and 160 for testing) spoken by four speakers in a terminal room environment. The test data of 160 sentences were recorded on two separate occasions (80 sentences on each occasion); the first (SET 1) was recorded during the same session as the 80 training sentences and the second (SET 2) was recorded in the same environment but five months after the training data. The syntactic grammar used for testing the system is a desk calculator task (sometimes called "voice programming") that contains 37 words. The performance characteristics of the Harpy system on the test data is given below.

The HARPY Speech Recognition System

Bruce T. Lowerre

April, 1976

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Submitted to Carnegie-Mellon University in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (Contract F44620-73-C-0074) and monitored by the Air Force Office of Scientific Research.

ib

Abstract

The Harpy connected speech recognition system is the result of an attempt to understand the relative importance of various design choices of two earlier speech recognition systems developed at Carnegie-Mellon University: The Hearsay-I system and the Dragon system. Knowledge is represented in the Hearsay-I system as procedures and in the Dragon system as a Markov network with a-priori transition probabilities between states. Hearsay-I uses a best-first search strategy of the syntactic paths while Dragon searches all the possible syntactic (and acoustic) paths through the network in parallel to determine the globally optimal path. Hearsay-I uses segmentation and labeling to reduce the effective utterance length while Dragon is a segmentation-free system. Systematic performance analysis of various design choices of these two systems resulted in the HARPY system, in which knowledge is represented as a finite state transition network but without the a-priori transition probabilities. Harpy searches only a few "best" syntactic (and acoustic) paths in parallel to determine the optimal path, and uses segmentation to effectively reduce the utterance length, thereby reducing the number of state probability updates that must be done.

Several new heuristics have been added to the HARPY system to improve its performance and speed: detection of common sub-nets and collapsing them to reduce overall network size and complexity, eliminating the need for doing an acoustic match for all phonemic types at every time sample, and semi-automatic techniques for learning the lexical representations (that are needed for a steady-state system of this type) and the phonemic templates from training data, thus automatically accounting for the commonly occurring intra-word coarticulation and juncture phenomena. Inter-word phenomena are handled by the use of juncture rules which are applied at network generation time, thereby eliminating the need for repetitive and time consuming application of phonological rules during the recognition phase. State transition probabilities are calculated dynamically during the recognition phase from speech dependent knowledge rather than a-priori from statistical measurements.

The system was tested on four sets of data containing 240 sentences (80 for training and 160 for testing) spoken by four speakers in a terminal room environment. The test data of 160 sentences were recorded on two separate occasions (80 sentences on each occasion); the first (SET 1) was recorded during the same session as the 80 training sentences and the second (SET 2) was recorded in the same environment but five months after the training data. The syntactic grammar used for testing the system is a desk calculator task (sometimes called "voice programming") that contains 37 words. The performance characteristics of the Harpy system on the test data is given below.

	WITH SYNTAX (BRANCHING FACTOR = 10.8)	WITHOUT SYNTAX (BRANCHING FACTOR = 37)
WORD ACCURACY		
SET 1	97.5%	90.8%
SET 2	93.1%	83.5%
SENTENCE ACCURACY		
SET 1	88.6%	70.0%
SET 2	70.9%	36.7%
RECOGNITION TIME (x REAL-TIME)		
SET 1	13.1	18.0
SET 2	13.2	18.0
RECOGNITION TIME (MILLION OF INSTRUCTIONS EXECUTED PER SECOND OF SPEECH)		
SET 1	4.5	6.2
SET 2	4.5	6.2

"SENTENCE ACCURACY" is the percent sentences that were correctly recognized (i.e., all words within the sentence were correctly recognized). "BRANCHING FACTOR" is the average number of words from which the system must choose (as determined by the syntax) at any point during the recognition process. The average number of words per sentence is 5.5, the average number of phones per word is about 7, and the average number of phones per sentence is about 39.

To the best of my knowledge, this represents the best performance of any system reported to date along the above dimensions of number of speakers, branching factor, accuracy, and time.

Acknowledgements

I wish foremost to thank my advisor Professor Raj Reddy who has been colleague, inspiration, and friend during my graduate studies. I would like to thank the members of my thesis committee, Professors Peter Andrews and Nico Habermann, and especially Doctor Lee Erman for their helpful and constructive criticism of this dissertation. I also wish to thank my wife Emily, whose patience has sometimes been strained, for moral support during the too many years of my graduate work.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Motivation

The original title of this thesis was "A Comparative Performance Analysis of Speech Understanding Systems". It was intended to be a study of the various types and levels of knowledge that are used in speech understanding systems (e.g., parametric schemes, acoustic-phonetic evaluation functions, and search strategies) to determine their relative strengths and weaknesses. One of the objectives of the original thesis was, after the analysis study, to combine the "good" features of the systems that were studied to form a system that would have performance characteristics superior to the others. However, the work on this "hybrid" system proved to be more interesting than the original analysis study.

The Harpy system evolved from a systematic study of several design choices of two speech systems previously developed at Carnegie-Mellon University: The Hearsay-I system (see Erman (1974)) and the Dragon system (see Baker (1975)). These two systems are of interest because of their contrasting features: model, structure, performance, and representation of knowledge. The following table shows some of these features:

FEATURE	HEARSAY-I	DRAGON
Model	cooperating parallel processes	probabilistic function of a Markov process
knowledge representation	procedures	Markov network
units of speech signal representation	segmentation	10 milli-second sample
search strategy	best-first	all paths in parallel
search time	8-50 times real-time	45-200 times real-time
search time variation	extremely large	near zero
performance	does not always find optimum recognition path. Occasionally results in no recognition	always finds optimum recognition path

syntax	uses anti-productions of a BNF grammar to generate the syntactic paths to search	syntax is an integral structure of the Markov network
phonetics	uses a phonetic dictionary and many "hard-wired" heuristics to do the phonetic parsing	phonetic pronunciations are an integral structure of the Markov network. contains no other phonetic knowledge
acoustics	uses user dependent phonetic templates and many "hard-wired" heuristics for evaluating the acoustic signal	uses user dependent phonetic templates and a simple probabilistic function for evaluating the acoustic signal

These design choices were carefully studied to determine their relative strengths and weaknesses. The criteria for evaluating these design choices were along the dimensions of (in order of importance) accuracy, speed, and size.

1.2 Goals of the Harpy System

The goals of the Harpy system are to combine the "best" features of the above two systems with additional heuristics to form a high speed and high performance system. The features of the Harpy system are:

FEATURE	HARPY
Model	Dynamic programming system
knowledge representation	state transition network
units of speech signal representation	segmentation
search strategy	a "few best" paths in parallel
search time	about 10-20 times real-time

search time variation	near zero
performance	nearly always (i.e., more than 99% of the utterances) finds the optimum recognition path
syntax	syntax is an integral structure of the state transition network.
phonetics	Phonetic pronunciations are an integral structure of the start transition network. Word juncture phenomena are also an integral structure of the network. Contains no other phonetic knowledge.
acoustics	Uses user dependent phonetic templates and a simple probabilistic function for evaluating the acoustic signal. Also contains phonetic duration heuristics for calculating state transition probabilities.

Several heuristics and techniques were developed to reduce the network size and complexity (see chapter 5). Also, new techniques were developed for the semi-automatic generation (and tuning) of the lexicon and templates that are needed for this type of "steady-state" system. This tuning has resulted in a vast improvement of the accuracy of the system.

1.3 Methods and Materials

The digital computer used in this study is a PDP KA-10, a 36 bit machine, which has an effective speed of about 342,000 instructions per second. The hardware speech interface to the computer consists of a zero crossing counting device (called a ZCC) and an analog to digital conversion device (called an ADC). The ZCC consists of five octave band filters (200-400, 400-800, 800-1600, 1600-3200, 3200-6400 Hertz) and an unfiltered band. The speech signal is sampled every 10 milli-seconds and six pairs of nine bit numbers are produced (one pair for each of the six bands): the maximum peak to peak amplitude and the zero crossing count for each sample. The ADC device produces nine bit samples of the speech signal every 100 micro-seconds. See Erman (1974) for a complete description of these devices.

The Dragon system and the Hearsay-I system both used the ZCC parametric data for recognition. An early version of the Harpy system also used this same parametric scheme but was later changed to the more accurate Itakura metric,[1] which utilizes the ADC data.

The speech data used in this study fall roughly into two broad categories: the original data used by Baker for his Dragon system and data recorded specifically for the Harpy system. Within both categories are two classes of data: training and test. Dragon's[1] original training data contained several utterances taken from each of five different task domains. This test data consisted of five sets, one set from each of these five task domains. All six data sets were recorded by the same speaker. The data used for the Harpy system were recorded by four separate speakers and contain the same 20 sentences in each set. Each speaker recorded three sets of these 20 sentences: one training set, one test set that was recorded during the same session as the training set, and one test set that was recorded five months after the training set. The following is a summary of the speech data sets used in this study:

DATA	TASK	LEX-SIZE	SPEAKER	COMMENT
BTRAIN	all	802	JB	Training for next five sets
BCHS	CHESS	24	JB	Chess game
BDOC	DOCTOR	66	JB	Medical diagnosis
BDSC	DESC CALC	37	JB	Desk calculator
BAPT	NEWS	28	JB	News retrieval
BFRM	FORMANT	194	JB	Formant tracking
BL1	DESK CALC	37	BL	Training
KP1	DESK CALC	37	KP	Training
DS1	DESK CALC	37	DS	Training
RG1	DESK CALC	37	RG	Training
BL2	DESK CALC	37	BL	Test, recorded with BL1
KP2	DESK CALC	37	KP	Test, recorded with KP1
DS2	DESK CALC	37	DS	Test, recorded with DS1
RG2	DESK CALC	37	RG	Test, recorded with RG1
BL3	DESK CALC	37	BL	Test, recorded 5 months after BL1
KP3	DESK CALC	37	KP	Test, recorded 5 months after KP1
DS3	DESK CALC	37	DS	Test, recorded 5 months after DS1
RG3	DESK CALC	37	RG	Test, recorded 5 months after RG1

The Formant task, while it is the most complex of the five tasks, is too large for the convenient testing of the new training techniques developed for the Harpy system. The Desk Calc system was chosen because it contains a small number of words but still contains non-trivial acoustic ambiguities. For example, the acoustically confusable words "alpha", "beta", "gamma", and "delta" all occur in syntactically equivalent positions. See Goodman (1976) for a complete treatise on the complexities of these grammars. The training data were used solely for the purpose of tuning the system and the test data were used solely for the purpose of testing the performance of the

[1]See Itakura (1975), and Markel, et al. (1973).

[1]See Baker (1975).

system after tuning. The errors made by the system on the test data were never analyzed nor specifically corrected; the accuracy of the system on the test data represents the true effectiveness of the tuning on the training data.

1.4 Outline of the Dissertation Presentation

Chapter 2 contains, besides a cursory description of the Hearsay-I system (see Erman (1974) for a complete description of thes system), an examination of the interesting features of the Hearsay-I system. The merits of these features (heuristics, algorithms, etc.) are discussed in order to gain some insight as to the strong and weak parts of the Hearsay-I system. Some performance results are given to form a basis of comparison among the three systems discussed in this thesis.

Chapter 3 contains an examination (similar to chapter 2) of the Dragon system.

Chapter 4 contains a detailed description of the Harpy system. This chapter discusses the implementation of the data dependent transition probability heuristic, implementation of the segmentation heuristic, and the motivation and implementation of the search space reduction heuristics. The problem with tuning of this type of system is discussed along with the actual scheme used. Performance results for the Harpy system are also given in this chapter.

Chapter 5 discusses the network data representation of knowledge that is used in this type of speech recognition system. Some network size reduction heuristics are also discussed in this chapter.

Chapter 6 discusses conclusions of this work and gives some insight as to what future work must be done to implement the Harpy system on a larger sized grammar.

2 FEATURES OF THE HEARSAY-I SYSTEM

2.1 Introduction

The Hearsay-I system is the result of an attempt to represent the diverse sources of knowledge needed for speech recognition as procedures. The system is highly modularized into separate knowledge source procedures. This modularity allows each knowledge procedure to contain highly specialized heuristics about its own knowledge. The Hearsay-I system is the product of about 10 man-years of effort from about five principal contributors. The author is responsible for adding a large amount of the acoustic-phonetic knowledge to this system. Historically, this was the first system to be demonstrated live.

2.2 Model

The model of the Hearsay-I system is one of parallel cooperating processes. The system contains three of these parallel processes (modules): semantics, syntax, and acoustics. However, for the purposes of comnarison with Harpy, only syntax and acoustics are considered. The search strategy used in the Hearsay-I system is a best-first search of the legal syntactic paths (at the word level) with backtracking. The search is driven by syntactic anti-productions which hypothesize words. The acoustics module verifies these words by either rating them or rejecting them. The search of the syntactic paths is done in a best-first strategy. See Reddy et al.(1973) and Erman (1974) for a detailed example and description of this system.

2.3 Representation of Knowledge

Knowledge in the Hearsay-I system is represented in procedural form. Complete details of the structure and recognition process of this system can be found in Erman (1974) and Neely (1973). The following is a brief description of the main modules and the recognition process of this system. The recognition portion of the system used in this study (i.e., without semantics) contains three main modules: the recognition overlord (called Rover), the syntax module, and the acoustics module. Rover is the coordinator for the syntactic and acoustic processes and contains all global data used by the other two modules. Rover is also responsible for the selection of the "best" syntactic path to follow. The "best" path is chosen from a list of all current partially completed syntactic paths. The recognition proceeds in a left-to-right fashion at the word level. A partially completed path contains all the words recognized so far and an indication of where the unrecognized portion starts. A typical partial path, from the desk calc task, may look something like "STORE FIVE PLUS THREE FOUR <FILLER>" where <FILLER> is the unrecognized portion (This might, for example, be the first part in the sentence "STORE FIVE PLUS THREE FOUR INTO ALPHA" of the Desk Calc task). The "best" path is defined as the partial path with the highest provisional rating. The provisional rating for a partial path is obtained by averaging the ratings of all words currently recognized in the path. Rover, after selecting the best path to follow, calls on the syntax module to hypothesize option words; these are all of the next words possible in the <FILLER> position. In the above example, syntax uses its anti-productions[1] of the BNF grammar to hypothesize the ten digits plus the words "IN" and "INTO". After receiving this list of option words from syntax, Rover calls on the acoustics module for verification and rating of these option words. Acoustics looks up each option word in its phonetic dictionary and maps[2] this phonetic spelling onto the parametric representation of the acoustic signal, taking notice of possible word juncture phenomena with the previous word (i.e., the word "FOUR"). After the phonetic mapping is done, acoustics rates each phone of the spelling against the actual acoustic signal. A total word rating is passed back to Rover from the combined ratings of all the phones in the spelling. Rover expands each option word with the words already recognized to form new partially completed utterances, and the rating given to each new partial path is composed of all the words currently recognized in the path (including the new word). Rover then sorts all the partial paths that its currently has and selects the highest rated one as the best one for the next recognition cycle. This searching continues until syntax hypothesizes and acoustics verifies the end of utterance for one of the partial paths.

The knowledge of the syntax module is represented as a BNF grammar specification. This BNF is used to generate the anti-productions for the hypothesis of words for partially completed paths. The BNF is also used to do a top-down parse of completed paths to verify the validity of an utterance. This complete top-down parse is needed to eliminate possible incorrect syntactic paths caused by the use of only two left context words for the anti-productions. For example, if a partial path is "STORE FIVE PLUS THREE FOUR <FILLER>", then the anti-productions, which look only at the words "THREE FOUR", will hypothesize, besides the correct words, the incorrect words "]" (end of sentence), "PLUS", "TIMES", "POWER", etc.

[1]The syntax module uses the two words immediately left of the <FILLER> part as context for determining what words can follow. The lists of left context words are called anti-productions (see Neely (1973) for a complete description).

[2]This process is also called "phonetic parsing".

The acoustic knowledge of the system includes a phonetic dictionary of lexical words in the grammar,[1] a set of user dependent acoustic-phonetic templates (called PP's),[2] and a large amount of explicitly programmed acoustic-phonetic knowledge for segmenting and labeling the acoustic signal, mapping the words to the segmented signal, and rating the mapped words (by use of the PP's and phonetic spellings) with the segmented signal. This "hard-wired" knowledge takes the form of heuristics that represent most of the actual code in the acoustic module. The heuristics deal with such phenomena as inter and intra-word junctures, coarticulation effects, acoustic realizations of sounds, etc.

2.4 System Tuning

System tuning for Hearsay-I consists of altering one or more of the following: the user dependent templates, the phonetic dictionary, or the speech heuristics in the acoustics module. The explicitly programmed speech heuristics are user-independent and represent general acoustic-phonetic knowledge. The phonetic dictionary represents dialectal knowledge and is changed only when dealing with a new dialect or when encountering new pronunciations. The PP's represent user dependent vocal characteristics and are usually not changed for an individual speaker once produced. The tuning of the PP's, for a new speaker, can be done by a pre-processing tuning program: The program requests the speaker to say a set of words that is used for extracting the PP parameters. The PP's can also be tuned dynamically by the Hearsay-I system itself: the system can be forced to recognize the correct path (by manually forcing Rover to select the correct path at every recognition cycle) and the acoustics module can alter the PP's according to the mapping of the correct path.

2.5 Performance Results

Following are the performance results of Hearsay-I (without semantics) on the five sets of Baker's data:

[1] There may be multiple entries per word.

[2] There is only one template entry per phone.

TASK	LEX	B-F	%WORDS	%UTTS	TIME	S.D.
CHESS	24	7.3	69	32	13.7	2.6
DOCTOR	66	3.7	49	24	9.4	3.8
DESK CALC	37	10.8	53	22	15.5	9.4
NEWS	28	2.7	74	50	10.8	6.4
FORMANT	194	3.8	33	33	44.4	23.5

LEX is the number of words in the grammar, B-F is the average branching factor for the grammar (the average number of words that may legally follow any word in a partial sentence of the language), %WORDS are the % words correctly recognized, %UTTS are the % utterances correctly recognized (all words within the utterance were correctly recognized), TIME is the number of times real-time needed for recognition, and S.D. is the standard deviation of TIME.

2.6 Strengths and Weaknesses

A powerful feature of the Hearsay-I system is its best-first strategy. This allows the system to follow one syntactic path to completion very quickly, as is shown above by the low recognition time needed.

Paradoxically, one of the major problems with Hearsay-I is its best-first search strategy, due to the need for backtracking. The obvious manifestation of this is the large variation in the time needed for recognition, as seen from the standard deviation. The actual variation would have been even worse were it not for an upper time limit imposed on the search. Because of this time limit cutoff, the system totally missed 18% of the utterances with no recognition at all. The variation in recognition time is due to the fact that the system searches only one syntactic path at a time. The total number of possible syntactic paths (sentences) in the grammar can range from vary large for a finite grammar to infinite for a recursive grammar. For example, the number of paths for the chess task grammar is more than 300,000,000. This combinatorial explosion of syntactic paths proves to be overwhelming, in a best-first search with backtracking, once the system loses the correct path.

Errors and incompleteness in the acoustic-phonetic knowledge aggravate the best-first search problem. There are many heuristics in the system that can completely reject a path; if a correct path is rejected due to an error in one of these heuristics, then there is no hope of recovery.

A good heuristic that is used to reduce the effective data size is the segmentation of the speech signal. Each 10 milli-second sample of the speech signal is labeled with the PP that best matches to it. Neighboring 10 milli-second samples with the same (or similar) PP labels are combined together into segments. These labeled segments are categorized into one of three types: sonorant, silence, or frication. Max-min detection of the speech signal amplitude is also used to segment syllable boundaries within the sonorant segments. Since the segmentation is done at a gross level (silence, frication, and sonorant), the ratio of segments to the number of 10 milli-second time samples is small. In one typical utterance of length 230 centi-seconds, there were 18 segments produced (the average segment being about 13 centi-seconds in length). The resulting segments are used as a guide for the phonetic mapping routine. Segmentation effectively reduces the amount of acoustic data that must be examined during the mapping and rating processes.

A weak feature of the system is that it does not remember what it has learned during the search of one path in order to apply it to another path. For example, assume the system is running on the chess task with semantics turned on.[1] Assume also that the sentence actually spoken was "Pawn to queen four". If the word "knight" gets a rating better than "Pawn" during the first recognition cycle, then the partially completed utterance that the system will work on during the next cycle is "Knight <filler>". Now assume that the system correctly recognizes the next two words. This will make "Knight to queen <filler>" the partial utterance to be considered for the next cycle. At this point a conflict will arise within the knowledge sources. Acoustics and syntax will agree on "four" as the next word but semantics will reject this word. Rover will, at this point, abandon this path completely and choose the next best path (presumably "Pawn <filler>") to follow. The system will now duplicate its effort in hypothesizing and mapping the same next three words. It does not save any information from one path to be used in the next. This is true for both positive information (words successfully identified) and negative information (words rejected).

2.7 Discussion

The features of the Hearsay-I system that were considered in designing the Harpy system are the following:

1) Segmentation of the acoustic signal can effectively reduce the amount of speech data searched.

[1]There is no loss of generality to the example to assume that semantics is turned on. It simply makes the example easier to understand.

2) A fast recognition can be achieved by searching a small number of paths (e.g., one in the case of the best-first search).

3) Backtracking while searching can be costly, especially in a large search space.

4) Duplication of effort should be avoided (i.e., knowledge gained from past searching should be used to guide future searching).

5) Heuristic speech knowledge is a useful guide to mapping and rating words.

3 FEATURES OF THE DRAGON SYSTEM

3.1 Introduction

The Dragon system was essentially a one-man effort and represents about two years of work. The system is interesting because of its simplicity of design, its mathematical tractability, and its high accuracy performance. The program code contains almost no speech dependent heuristics. The author is responsible for tuning this system to the point where it is the first system developed at Carnegie-Mellon University to achieve 100% accuracy on a set of speech data (see QTRAIN experiment in section 4.8.1) using only syntactic and acoustic-phonetic knowledge.

3.2 Model

The model of the Dragon system (See Baker (1975)) is one of a probabilistic function of a Markov process. The system achieves recognition by updating state probabilities of a Markov network on every 10 milli-second speech sample. The network contains all the syntactic and phonetic knowledge used in the system. Inter-state connections are indicated by a-priori transition probabilities. The network also contains intra-state a-priori transition probabilities.

3.3 Representation of Knowledge

The major knowledge sources for the system are: the network which contains the integrated knowledge of both syntax and lexical phonetic spellings, a set of user dependent acoustic-phonetic templates, and an acoustic-phonetic probability[1] matching

[1]The word "likelihood" may be better suited here. The acoustic probabilities are numbers (one per template) between 0 and 1 that give the relative "probability" of each template matching to the actual acoustic sound present. The sum total is not necessarily equal to 1.

routine. The network consists of a set of states, where each state represents a phonetic sound within a lexeme, with connection pointers between them coupled with inter-state a-priori transition probabilities. Each state also contains an a-priori intra-state transition probability. There are two special states in the network called the "initial state" and the "final state". All paths that lead from the initial state to the final state (via intermediate states of course) represent not only all legal syntactic paths but also all phonetic pronunciations of these syntactic paths. See chapter 5 for a complete description of the networks.

The Dragon system uses a dynamic programming scheme to search all possible paths through the network in parallel by the following formula:

$$P_{i,t} = A_{i,t} \max_{j}(P_{j,t-1} T_{j,i})$$

$P_{i,t}$ is the probability of being in state i at time t, $A_{i,t}$ is the acoustic match probability for state i at time t, $T_{j,i}$ is the a-priori probability of transitioning from state j to state i,[1] and t is in 10ms time units and runs from 1 to L (the length of the utterance). The initial condition at t=0 is that all state probabilities are 0 except the initial state, which is 1. At each 10ms sample, the probability of being in each state is calculated by the above formula and the $P_{j,t-1}$'s that represents the $\text{Max}(P_{j,t-1}T_{j,i})$ are saved. These are the best prior states. If an utterance has length L and there are N network states, then the system must save LxN best prior states. After the state probabilities have been calculated for all time samples, the system backtracks through the best prior states to find the globally optimum path (called simply the "global path") in the following recursive way. The state on the global path at time L is the final state. The state on the global path at time t is the best prior state for the global path state at time t+1. The recognition result is taken from the syntactic path contained in the global path as it runs from the initial state to the final state.

There are two other types of network paths that need to be defined. A "correct path" is one whose contained syntactic path represents what was actually spoken. There is usually more than one correct path. The "global correct path" is the correct path with the highest probability. There is a possibility of there being more than one correct path with the same highest probability. The system achieves 100% correct recognition on an utterance only when the global path and the global correct path are the same.

[1]Note: j may equal i (intra-state transition). Most transition probabilities are 0 (i.e., there are only a few states into which any state may transition.

3.4 System Tuning

The Dragon systems contains no coded heuristics except for the acoustic probability matching routine. The inter-state and intra-state a-priori transition probabilities are fixed, as is the "standard" phonetic dictionary.[1] The only tuning that can be done for the Dragon system is the selection and modification of the acoustic-phonetic templates. Each template is an exemplar of an allophonic variation of the phone that it represents and is selected from a single 10 milli-second sample that best represents the allophonic group. The allophonic groups are obtained by hand labeling the training data and matching each 10 milli-second sample of the training data to the templates obtained so far. If a 10 milli-second sample exceeds a threshold of acceptance (probability of .1 was used by Dragon) for all templates (allophonic variations) of the desired phone, then a new template is added for this phone. Dragon uses 168 templates which cover 33 phones.

3.5 Performance Results

The following are the performance results of Dragon on the five sets of Baker's data:

TASK	LEX	B-F	%WORDS	%UTTS	TIME	S.D.
CHESS	24	7.3	94	68	48.0	0.6
DOCTOR	66	3.7	88	76	67.4	1.1
DESK CALC	37	10.8	63	17	83.1	1.0
NEWS	28	2.7	84	50	54.7	0.6
FORMANT	194	3.8	84	33	173.8	3.3

LEX is the number of words in the grammar, B-F is the average branching factor for the grammar, %WORDS are the % words correctly recognized, %UTTS are the % utterances correctly recognized (all words within the utterance were correctly recognized), TIME is the number of times real-time needed of for recognition, and S.D. is the standard deviation for TIME.

[1]This dictionary is, on rare occasions, manually changed to correct gross errors that are detected in it. One example of this was to change the phonetic spelling of "queen" from "K W IY N" TO "WH IY N".

3.6 Strengths and Weaknesses

The strongest feature of the Dragon system is that is searches all possible paths in parallel. This always allows it to achieve a recognition since it never gets lost in the combinatorial explosion of syntactic paths, always find the globally optimum path and, run at a constant speed, as shown by the near zero S.D. above. A timing model for the Dragon system in number of times real-time is:

(recognition time) = 20.9 + .067(#network states)

The weaknesses of the Dragon system are: 1) the large amount of computation time needed for recognition. This high computation time is due to the fact that Dragon updates all the state probabilities at every 10 milli-second time sample. The system must also perform an acoustic match of all its phonetic templates (168 of them) at every time sample. 2) the dictionary used by Dragon contains mostly "phonemic" spellings. That is, the spellings contain phones that represent more than a single acoustic type of sound. For example, burst phones such as /T/ are sometimes used to indicate both the silence part and the aspiration part of the sound. The problem with the phonemic dictionary stems from the fact that Dragon is unable to recognize dynamic changes in the speech signal due to its myopic view of one 10ms sample per state probability update. A phonemic dictionary does not allow for the "steady-state" nature of the search algorithm (see section 4.8.1). 3) there is no provision for word juncture phenomena. 4) there are multiple template entries per phone (168 templates covering 33 phones)[1] used in an attempt to compensate for deficiencies 2) and 3). 5) (a technicality) the floating point hardware limitations of the machine used are such that the state probabilities would underflow within about 30 time samples. This necessitated a time-consuming scaling operation.

The state transition probabilities used by Dragon are a-priori constants. Its intra-state transition probabilities are .9 (or 0) and its inter-state transition probabilities are .1 divided by the branching factor for that state (i.e., the number of inter-state transitions). An accidental discovery led to the following experiment: On one set of test data, all intra-state probabilities were set to 1 and all inter-state probabilities were varied with the following results:

[1]The system chooses the best template match per phone for the acoustic match probability.

inter-state probability	%Word Accuracy
1.0	66.0
.5	88.1
.3	93.2
.2	98.3
.1	100.0
.01	100.0
.001	96.6

The inter-state transition probability appears to act as a "throttle" on the number of words recognized in an utterance. For a probability greater than .1, the number of words in the global path are more than the number of words actually spoken; below .01, the number in the global path are fewer than actually spoken. The system achieved 100% correct recognition for all probability settings between .01 and .1. The reason for this behavior is that the real "action" of the recognition process is due to the acoustic match probabilities rather than the transition probabilities.[1]

One reason for errors in the Dragon system is the large number of templates used compared to the small number of phones that they represent. This results in a large number of overlaps within the acoustic-phonetic space. These overlaps create ambiguities within the phonetic spellings. For example, Dragon uses three templates to represent the silence phone "-"; only one of these is true silence. The other two are are noise like sounds that occur at sonorant to silence transitions (voice bars, etc.). These noise like transitions match very closely to the nasal template which results in ambiguities between nasals and silence.

3.7 Discussion

The Dragon systems demonstrates that it is possible to build a connected speech recognition system that achieves better than 85% word accuracy using only syntactic and acoustic knowledge. More importantly, the system also shows that there is a search algorithm (that does no backtracking) which guarantees a recognition and in a deterministic amount of time.

The most significant feature of the Dragon system, as compared to most other current speech recognition systems, is its almost total lack of speech-dependent

[1] See section 4.4.

heuristic knowledge. Dragon treats speech recognition as a mathematical computation problem rather than as an artificial intelligence problem.

The features of the Dragon system that were considered in designing the Harpy system are the following:

1) The network data representation is a tractable means of representing the combined knowledge of syntax and the phonetic dictionary spellings.

2) The dynamic programming scheme for searching all the network paths in parallel guarantees not only that a recognition will always be achieved (and in a deterministic amount of time) but also that the recognition will be the globally optimum one (with the given model).

3) Searching all paths (even in parallel) can be time consuming.

4 THE HARPY SYSTEM

4.1 Introduction

The Harpy system is an attempt to combine the best features of the Hearsay-I system and the Dragon system. Harpy contains a mathematically tractable model, as in the Dragon system, plus speech-dependent heuristics, as in the Hearsay-I system, for better performance and speed than either of the other two systems. Harpy also incorporates many new techniques (and heuristics) for increasing its speed and accuracy. Also, an entirely new semi-automatic system was designed to generate the lexicon and templates needed for Harpy. The following is a comparison of the two previous systems and Harpy:

	HEARSAY-I	DRAGON	HARPY
Knowledge Representation	Procedural embedding	Markov networks	Transition networks (No a-priori transition probabilities)
Search strategy	best first with backtracking	all paths in parallel with no backtracking	"best few" in parallel with no backtracking
Segmentation	yes	no	yes
Phonetic classification	procedural	multiple templates	unique templates
Word Juncture Knowledge	procedural	none	integral part of network

4.2 Model

The model of the Harpy system is one of a dynamic programming system with the use of heuristics to reduce the search space, thereby increasing its speed. The system uses a network that represents not only all legal syntactic paths, but also all pronunciations of these legal paths. The combined knowledge of syntax and lexicon is contained within all the legal "syntactic-phonetic" paths of the network. The network contains inter-state connections, but no a-priori transition probabilities.

4.3 Representation of Knowledge

The majority of the knowledge in the Harpy system is represented within the network. The network is generated by a network compiler from the BNF grammar specification, the phonetic dictionary of the lexical words, and the inter-word juncture rules. Thus, the network is a representation of these three knowledge sources. As in the previously discussed systems, Harpy uses a set of speaker-dependent acoustic-phonetic templates to represent the acoustic realization of the phones in the dictionary.

4.4 Data Dependent Transition Probabilities

One of the weaknesses of the Dragon system is its inability to perform an accurate acoustic mapping. The reason for this is due to its a-priori constant inter-state and intra-state transition probabilities. A global path of length P (time samples) that contains S number of states has a total of S-1 inter-state transitions and P-S+1 intra-state transitions[1] regardless of how the states are mapped to the acoustic signal. Since the transition probabilities in Dragon are a-priori constants, the "penalty" paid for making all the necessary state transitions is the same for all possible mappings of the global path. The result is, that given the constant intra- and inter-state transition probabilities, the acoustic mapping is determined solely by the acoustic match probabilities. The acoustic knowledge of Dragon is errorfull and incomplete due to errors in the templates[2] and dictionary. These deficiencies cause Dragon to have the

[1]The initial state is assumed to be at time sample 0.

[2]Template errors are of three types: a missing template, an extraneous template, or a template that is not representative of its phonetic group (a "mis-tuned" template).

tendency to either "over-parse" or "under-parse" the phone of a state. Over-parsing occurs when a particular state phone incorrectly receives a better rating than it should for a large number of connected time samples.[1] This can be very severe if the phone in question is a burst such as "B" or "P" and it gets parsed for 100msec or more. Under-parsing is the opposite of over-parsing. An example of under-parsing is a stressed vowel being parsed for one time sample (10msec).[2] Because Dragon is able to "hide" the bad acoustic probabilities by this time warping of the mapping, it can produce some very erroneous mappings and, therefore, incorrect recognitions.

Harpy attempts to overcome this deficiency by using data-dependent transition probabilities instead of the a-priori constant transition probabilities. The idea is to heuristicaly calculate the transition probabilities for each state during the recognition process from speech-dependent knowledge, such as pitch, amplitude, intrinsic phonemic durations, etc. Currently the system uses only the intrinsic phonemic durations. Harpy uses a file of minimum and maximum expected durations for each phone (see appendix E) and the current durations of the network states to calculate the transition probabilities. The current duration for a state is the number of time samples for which the state was its own best prior state.[3] The inter-state transition probability from a state is 1 if the current duration for that state is not less than the minimum expected duration for the phone of that state. If the current duration is less than the minimum expected, then the inter-state transition probability from that state is reduced by an heuristic amount that is proportional to the difference between the minimum expected and the actual current duration. If I is the the minimum expected duration, C the current duration (with C<I) then the inter-state transition probability is H^{I-C} where H is a heuristic value. Similarly, if the current duration is not greater than the maximum expected duration, then the intra-state transition probability is 1, otherwise it is H^{C-A} where A is the maximum expected duration and C and H are as before. The current value used for H is .1. A value of 0 for H would be tantamount to a total rejection of a path should the state duration fall outside the interval between the minimum and maximum expected durations. A large value for H (close to 1) and a small interval between the minimum and maximum expected durations would produce a Gaussian like distribution for the transition probabilities.

There is clearly much room here for experimentation with the H value, maximum and minimum durations, and other sources of knowledge (e.g., pitch and amplitude) that can be applied in calculating the data-dependent state transition probabilities. However, as was shown with the Dragon system (see chapter 3), the transition probabilities have only a secondary effect on the recognition results. The simple-minded scheme shown above is sufficient to produce very reasonable results and to

[1]This error is usually caused when there is more that one template for the given phone and at least one of the extra templates is not representative of the phonemic sound.

[2]This type of error is usually caused by a mis-tuned template or a missing template.

[3]i.e. the number of intra-state transitions.

eliminate the gross mapping errors. A typical result of using the data-dependent transition probabilities is 96.2% word accuracy and 89.5% utterance accuracy contrasted with 84.8% word and 57.9% utterance accuracies using the a-priori constant transition probabilities.

It is important when using the data-dependent transition probabilities that the phonetic dictionary contain no extra phones. Dictionary errors of this type are handled easily by Dragon by under-parsing. However, extra phones have a severe effect on the accuracy of the Harpy system since the system is forced to parse each phone for its minimum duration.

4.5 Optimization

The first step in designing the high speed Harpy system was to optimize the Dragon system. The optimized Dragon system served as the skeleton for the Harpy system. The optimizations done to Dragon were the reduction of program size, code streamlining, and improvements in the algorithms used. The information that must be saved during the recognition process is the best prior state for each state per time sample. The maximum number of time samples allowed is typically 800 (10ms time samples for 8 seconds of utterance). Large networks may contain 2500 or more states. This makes a total of at least 2,000,000 states that must be remembered during the forward trace through the network. This is greater than the main memory capacity of the machine used. Dragon uses a clever scheme to reduce the memory capacity needed for saving the best prior states; it takes advantage of the fact that each state has only a small number of possible prior states and, in fact, most states have only one possible prior state.[1] Dragon uses only as many bits as are necessary to uniquely specify the best prior state for each state. This greately reduces the storage necessary to save the best prior-state pointers to about 80,000 words of memory. Because this still will not work for very large networks and since the Harpy system is larger (see 4.10), Harpy does not save all the best prior states in main memory at one time. Harpy saves only a small number of prior states (typically 100 time samples worth) in memory at a time and when more room is needed, they are saved on external storage (a high speed drum).

The code, although left in SAIL,[2] was streamlined by combining redundant loops,

[1]Any state may undergo an implicit intra-state transition. "Prior" state here implies an explicit inter-state transition.

[2]The program code is written in an Algol-60 based language called SAIL (see VanLehn (1973)).

eliminating small loops by doing the operations linearly, and eliminating as much array subscripting as was possible.

The most significant change in the algorithms was to replace the probabilities by their logs and thus do additions instead of multiplications. The big savings here is not that addition is faster than multiplying but rather that the need to scale was eliminated. The worse acoustic match probability that is allowed is .01. Since the machine hardware does not accommodate a number smaller than about 10^{-39}, the exponents could underflow within about 30 time samples. Thus the time consuming scale checking and updating was eliminated by using log probabilities.

The formulas for the recognition times (in number of times real-time) for the original and optimized Dragon systems are:

original	(rec.time) = 20.9 + .067(net size)
optimized	(rec.time) = 18.8 + .0305(net size)

The optimized Dragon system was used as the basis for the Harpy system. All code that was added to the Harpy system was intentionally streamlined for speed.

4.6 Segmentation

Reduction of the number of time samples used to update the state probabilities is accomplished by segmentation of the input data into larger sized units than the 10 milli-second time samples. The only critical problem involved with doing segmentation for this type of recognition system is that there must be <u>no</u> missing segments. The system will map one or more time samples to one network state but it will never map more than one state to a single time sample. If the global path for an utterance of U 10 milli-second time samples has P states from initial state to final state ($P \leq U$), then a segmentation scheme may produce anywhere between P and U number of segments. If it produces fewer than P segments, the global path cannot be parsed and therefore will not be recognized. The fewer segments that are produced (but greater than P, of course), the faster will be the recognition speed. One measure of efficiency of a segmenter is how few segments it produces. However, any errors that it makes must be on the side of too many rather than too few segments.

Another point to be considered is that the time needed to do the segmentation must not outweigh the time saved during the recognition process. The segmenter adds another overhead to the recognition time, usually proportional to the length of the utterance. For example, it is possible to build a segmenter that will produce perfect segmentation.[1] However, the time needed to produce the segments will far outweigh any advantage gained during the recognition process. The total efficiency of a segmenter should be a measure of the total time saved during the recognition process, rather than how few segments it produces.

An underlying assumption made so far about segmentation is that the recognition system will obtain the same accuracy using segments as it does without segments. Clearly this need not be true. The accuracy of the segmenter will affect the accuracy of the recognition system. An early "segmentation" experiment that was tried was to use every N'th 10 milli-second time sample. The following are the results of this experiment on one task:

N	WORDS CORRECT(%)	# TIMES REAL TIME
1	100.0	104.7
2	85.2	52.8
3	36.6	35.4
4	16.9	26.5

As expected, the time[2] needed is inversely proportional to N. However, the recognition accuracy suffers greatly by such a simple-minded segmentation scheme. Harpy is a "steady-state" system; that is, the phones used in its dictionary (and consequently in the network) must represent a "stable" portion of the acoustic signal. This is due to the fact that Harpy looks at only one segment, which must be composed of acoustically similar 10 milli-second speech samples, per state probability update. The scheme used above, while each segment is composed of similar type speech samples (since there is only one per segment), does not account for every different acoustic signal encountered since it ignores a major portion of the speech samples. Consequently, there is a great loss of acoustic information.

The accuracy of a segmenter can be determined by how close the recognition system's results are to the results obtained on the same data not using segments.[3] In fact, it is possible for the use of an "intelligent" segmenter to produce better results from the recognition system than can be achieved without the use of segments, by eliminating or smoothing the noise in the 10 milli-second samples. It is sufficient to say that the segmenter is accurate if the results produced with its segments from the recognition system are no worse than the results produced without segments.

[1] "Perfect" means the number of segments exactly equals the length of the global path. Such a segmenter can be obtained trivially by running the recognition process twice, using the parsing results of the first recognition as segments for the second run.

[2] The overhead for segmentation in this case is essentially 0.

[3] This assumes that there are no errors in the phonetic dictionary.

The use of segments requires a small change in the recognition algorithm. Since the segments produced are not of uniform length, the state acoustic-phonemic match probabilities and the intra-state transition probabilities must be weighted by the length of the current segment during the recognition process.

The segmentation algorithm used in the Harpy system works as follows: The 10 milli-second time samples are processed one at a time to extend the "current" segment. As each 10 milli-second time sample is processed. it is compared against the first 10 milli-second sample of the current segment and against the middle sample. The current segment is considered complete when the distance[1] between the current 10 milli-second sample and either the first sample or the middle sample exceeds an heuristic threshold; the current 10 milli-second sample then becomes the first sample of the next segment. The linear predictor coefficients that are to represent the now complete current segment are obtained from the sum of the autocorrelation coefficients of all the samples in the segment. The heuristic threshold setting, for segmentation, used in the Harpy system (currently .6) was determined experimentally. Increasing the threshold results in fewer segments being produced, while lowering the threshold results in more segments being produced. Ideally, the threshold should be as large as possible without combining acoustically dissimilar segments.

The ratio of 10 milli-second samples to segments that the segmenter produces is dependent on the acoustic signal. On the average, it is about 3.5. The overhead for the segmenter takes 1.3 times real-time (i.e., processing time is 13 milli-seconds per 10 milli-second sample). The segmentation scheme gives the Harpy system better than a three fold speed up. The speed up is more than expected (due only to the data reduction) because the search space is also reduced by constraining the number of possible paths that can be searched (see section 4.,). The following are some timing results of and old version of Harpy, which used the ZCC parametric scheme, with and without segmentation (in number of times real-time):

TASK	WITHOUT	WITH
CHESS	18.3	5.2
DOCTOR	28.1	9.2
NEWS	22.1	6.5
DESK CALC	57.4	17.0
FORMANT	104.7	34.4

The word accuracy is 99.0% for the system without segmentation and 94.3% with segmentation. The drop in accuracy was attributed to errors in the dictionary rather than errors in segmentation (see 4.8.1).

[1] "Distance" is the acoustic-phonemic metric that is used. In this case it is the negative value of the Itakura metric probability (Log) match between a pseudo template generated from the current 10 milli-second sample and one of the two samples in question.

4.7 Search Space Reduction

The speed up philosophy of the Harpy system is to use heuristics to reduce the calculations and search space as much as possible during the recognition process. To see where heuristics can be applied, it is necessary to look at the time formula shown in section 4.5 in more detail.

$$(\text{rec.time}) = 2.2 + .04(\#\ \text{templates}) + .027(\#\ \text{states}) + .005(\#\ \text{pointers})$$

"Rec.time" is number of times real-time, # pointers is the number of inter-state connections in the network. A more enlightening way to express the formula is:

$$(\text{rec.time in seconds}) = (\#\ \text{time samples})(.022 + .0004(\#\ \text{templates}) + .00027(\#\ \text{states}) + .00005(\#\ \text{pointers}))$$

This shows the four main areas where Harpy applies heuristics to reduce its calculations and search space: the number of time samples used, the number of templates used for the acoustic match probabilities per time sample, the number of states checked per time sample, and the number of pointers checked per time sample.

The methods used by Harpy to reduce the number of templates, states, and pointers checked per time sample are based on the notion that not all network state probabilities need be updated at every time sample. There is a simple justification for this. At time sample 0, there is probability 1 of being in the initial state and probability 0 of being in all other states. The only states that have non-zero probability at time sample 1 are those states that can be reached in one transition from the initial state. Similarly, the only states that have non-zero probability at time sample N are those states that can be reached in N transitions (both inter and intra-state) from the initial state. At M time samples from the end of the utterance, the only states that need be checked are those from which the final state can be reached in M transitions. Unfortunately, this significantly reduces the number of states that need to be checked only near the start or end of the utterance. Intuitively, the only paths that need be checked at every time sample are those that are "obvious", or conversely, the paths that are "obviously" wrong can be ignored. Harpy implements this notion of "obvious" by the use of heuristic thresholds for the states that are checked at every time sample. The only paths that Harpy follows are those whose states have "high" probability while those paths whose states have a "low" probability are ignored. "High" and "low" are defined by the heuristics used in the search reduction.

The number of pointers checked at every time sample is directly dependent on the number of states checked; thus, their number is reduced automatically by reducing the number of states checked. Dragon, at the start of every time sample, calculates the probabilities for all of the template matches since it uses them all at every time sample. Harpy reduces the number of templates checked at every time sample by calculating the template probabilities only on demand (if they have not already been calculated). Thus, the search space reduction (and time reduction) problem in Harpy becomes one of reducing the number of states (and therefore paths) that is checked at each time sample.

The heuristics and algorithms used by Harpy to reduce the number of paths that it follows are not speech-dependent and can be applied to any dynamic programming system of this type. Hearsay-I reduces its search space by doing a best-first search. That is, it searches one path as long as it looks most promising. If the expectation value for the current path falls below that of another path, then the current path is temporarily abandoned and the next best path is followed. This process continues until a path is completed with an acceptable value. An early experiment done with Harpy was to implement a best-first search strategy. This required a major effort[1] and was done only for the sake of completeness as a fair comparison with Hearsay-I. The following are the results of three systems on the Chess task: Harpy with segmentation (see 4.6) in best-first mode, Hearsay-I, and Harpy with segmentation but searching all paths (i.e., "Dragon mode").

SYSTEM	% WORDS	TIME
Harpy+BFS+segmentation	70	41
Hearsay-I	69	14
Harpy+segmentation(Dragon mode)	96	5.2

"TIME" is the number of times real-time and "% WORDS" is the percent of words correctly recognized. There was an attempt made to run the best-first Harpy at the 10 milli-second sample level. However, the system was manually aborted on the first utterance when it ran for more that 800 times real-time (about 30 minutes of CPU time) without a recognition. It is clear that the best-first search technique has severe limitations when the search space becomes large. It is the large number of confusable paths and the subsequent large amount of backtracking that causes the foundering of the best-first search algorithm.

[1]The entire search space (state probabilities for all states for each time sample) must be saved at all times. The search involves updating the state probabilities along the best path. Also, an ordered list of normalized probabilities for each partial path must be kept for selecting the best path to follow. This differs from the "normal" Harpy search mode in that normally only the state probabilities of the previous time sample need be kept for updating the current state probabilities. In best-first mode, all state probabilities for all time samples must be kept since the search can "jump" around.

To overcome these difficulties, Harpy searches only a few "best" paths in parallel. This is a compromise between searching all paths and searching only the best path. Here again, "best" is defined by the heuristics and algorithms used. The first heuristic tried in Harpy was to search only a fixed number (N) of paths (actually, states). This was done within Harpy by maintaining a current and previous list of best states. The only candidate states (potential states that may go into the current best list) that were checked were those states that could be reached in one transition (both inter and intra) from the states in the previous list.[1] At time sample 0, the only entry in the current list was the initial state. At every time sample after that, the current list became the previous list and the candidate states were sorted to determine the rank ordering of probabilities from best to worst and the "best" states were placed into the current list. There were two problems with this method. First, the candidate states had to be sorted to determine which belonged in the current list and which could be ignored. This sort added an overhead to the search time (about 30% of the total search time was devoted to sorting).[2] Second, and this was critical, each state that was on the global path at each time sample had to be within the N best states; otherwise the global path would be broken and therefore not recognized. A statistical study showed that for more than 50% of the time samples, the states that were on the global path had the best probability. However, on rare occasions (less than 1% of the time samples)[3] the probability of the global path states dropped to such a low level that more than 30% of the total network states had probabilities greater than or equal to the global path state. The time needed to search each "best" state by this scheme (not including sorting time) was about twice what it was when searching all states (i.e., Dragon mode) due primarily to the addition of forward pointers (see timing model at the end of this section). Therefore, there was no time advantage realized when the fixed number of states searched exceeded about 50% of the total number of network states. This figure dropped to about 30% when the overhead for the sorting scheme used was added. This meant that there was no time advantage for this system when the number of states searched was large enough to guarantee that the global path would not be broken. Looking at this another way, the minimum search time needed (assuming zero sort time) to guarantee that no global path would be broken was about 60% of the time needed to search all paths. The "payoff" in this case was deemed too small to warrant further investigation of different sorting strategies.

[1]To determine which states a particular state may transition into requires a list of forward pointers. Each state in the Harpy network thus has, in addition to the list of back pointers, a list of forward pointers. This is the only significant difference in the format of the networks as used by Dragon and Harpy.

[2]Since the states in the prior best list were in decending probability order, the candidate states which were produced from the best prior states were roughly in sequence by probability. The sorting scheme used took advantage of this fact by placing the candidate states in the current best list one at a time. The current candidate state was assumed to have a probability worse than all states already in the current best list and the current best list was searched in reverse order to find the proper place for the current state.

[3]The average utterance length was about 150 milli-seconds. Therefore, the probability was that there was at least one 10 milli-second time sample per utterance where this phenomena occured.

The time samples where the ranking of the global path falls to a low level are the places where many different paths become confusable with the global path. The current version of Harpy takes advantage of this phenomenon by searching a variable number of "best" paths in parallel. The "best" states are those states whose probability is within a threshold amount of the highest state probability at each time sample. Formally, the set B_t of best states is:

$$B_t = \{S_i \mid P_i > H_t \times T \quad \forall i\}$$

where S_i and P_i are state i and probability of state i respectively, H_t is the highest state probability at the current time sample, and T is the threshold.[1] The threshold setting is determined empirically. It should be as large as possible without excluding the global path states at any time sample. The following table shows the effect of different values for T. These data are from the Formant task which has 2356 network states.

T	WORD ACCURACY(%)	AVE. # OF STATES SEARCHED PER SEGMENT
1	0	1
10^{-1}	18	5
10^{-2}	37	9
10^{-3}	46	20
10^{-4}	66	33
10^{-5}	73	61
10^{-6}	86	93
10^{-7}	84	137
$10^{-8.5}$	84	216

For comparison, Dragon gets 84% word accuracy on this data set. At large threshold settings (close to one), the number of states searched per time sample is very small and the accuracy is poor because of the large number of global paths that are broken. As the threshold is lowered, more states are searched at every time sample and consequently more global paths are recognized. When the threshold is small enough, all the global paths are recognized and the accuracy is the same as for Dragon. The accuracy reaches its peak at this point and remains at this peak for all lower setting of the threshold.[2] This occurs at about 10^{-7}, which is the "standard" threshold setting

[1]Harpy uses log probabilities rather than actual probabilities. The set of best states defined by Harpy is $\{S_i \mid P_i > (H_t - T') \ \forall i\}$ where T' is the negative log of T.

[2]There is an interesting anomaly in the above table. The accuracy actually reaches a peak at a threshold setting of 10^{-6} and then lowers slightly and remains constant for all lower threshold settings. This is a quirk of this data set and is caused by a confusion between the global path and the globally correct path. There is one utterance in this data set that Dragon does not correctly recognize (i.e., the global path is not a correct path) but Harpy does correctly recognize at a threshold of 10^{-6}. At

chosen for Harpy.[1] This threshold has been tested on all five of Baker's data sets, and with it Harpy achieves the identical recognition results as Dragon.

Since each candidate state is now compared against only one other state (the highest state probability) at each time sample, the need for sorting the candidate states has been eliminated. This greatly increases the speed of this heuristic. There is, however, one small technical problem with this heuristic. The highest state probability at each time sample is not known until all the candidate states have been checked, since the highest state probability comes from one of the candidate states. Harpy could save all the current candidate states until the highest state probability is found, then place the appropriate candidate states in the current best list. However, a simpler and faster technique is used in the actual implementation of this heuristic. The best lists (both previous and current) are unsorted with one exception -- the first place in both lists is reserved for the highest probability state. At the start of each time sample, the highest state probability (found so far) is set to 0. As each candidate state is processed, its probability is checked against the current highest state probability using the threshold T to determine if it belongs in the current best list.[2] In addition, the candidate state's probability is checked directly against the current highest state probability. If the candidate states' probability is higher, this state is placed at the beginning of the current list (the state that was in first place is added to the end) and its probability becomes the current highest. In the majority of cases (greater than 90%), the current highest probability state will be one of the states following the previous highest probability state. Since the previous highest probability state, which is first in the previous list, is the first to be processed, the current highest state probability is usually found immediately. For the cases where the highest state probability is not immediately found, there may be some states placed in the current best list that do not meet the heuristic criterion. However, the difference between the highest state probability and the "pseudo" highest state probability that was produced from the previous highest is so small that the number of extraneous states placed in the current list is negligible (i.e., fewer than 0.1%).

Following are the results of Dragon and Harpy on Baker's five sets of data. These tests were done after an attempt was made to improve Dragon's performance by tuning the templates and dictionary (see sections 4.8 and 4.9). The accuracies are % words correct and the times are in number of times real-time. "Harpy states" is the mean number of network states per time sample that Harpy actually searches. "NO.POINTERS" is the number of inter-state connections in the network (i.e., either forward or back pointers).

---

this threshold setting, Harpy breaks the global path for this utterance, resulting in a correct path being recognized!

[1]This setting is for the ZCC parametric scheme. A value of 10^{-5} is used for the Itakura metric.

[2]The first candidate state checked will always be placed in the current best list.

TASK	DRAGON ACCURACY	HARPY ACCURACY
CHESS	100	96.2
DOCTOR	100	88.0
NEWS	100	90.8
DESK CALC.	94.8	89.7
FORMANT	100	90.1
Total	99.0	90.8

TASK	DRAGON TIME	HARPY TIME
CHESS	18.3	1.5
DOCTOR	28.1	2.5
NEWS	22.1	1.9
DESK CALC.	57.4	4.1
FORMANT	104.7	5.8

TASK	STATE SIZE	NO.POINTERS	HARPY STATES	(%)
CHESS	323	1020	24	7.4
DOCTOR	592	1147	35	5.9
NEWS	439	884	30	6.8
DESK CALC.	973	5040	66	6.8
FORMANT	2250	7630	80	3.6

83 templates were used in the above runs. Following is a timing model for the Harpy system using the ZCC parameters:

(Time needed in seconds per time sample) = .019 +
.00046(# templates checked per time sample) +
.00047(# states checked per time sample) +
.000096(# pointers checked per time sample).

The timing model for Harpy using the Itakura metric is:

(Time needed in seconds per time sample) = .019 +
.00071(# templates checked per time sample) +
.00047(# states checked per time sample) +
.000096(# pointers checked per time sample).

The above times must be multiplied by the total number of time samples, and the segmentation overhead (see 4.6) must be added to obtain the total processing time. The number of templates, states, and pointers checked per time sample is a function of both the heuristic setting and the confusibility among the network paths. The more confusable the network paths are, the closer their probabilities will be and, consequently, the larger the number of states that are placed in the best lists.

Comparing the above Harpy time model with Dragon's time model (see page 25) indicates the following: The reduction in overhead (.019 vs. .022) is due to some of the general calculations being done at specific times (e.g., the template acoustic match probabilities are done only when needed in Harpy); this is reflected in the rise of the template calculation time (.00046 vs. .00040). The increase in state calculation time (.00047 vs. .00027) is due to the processing overhead of the best lists. Finally, the doubling of the pointer processing is due to a list of following states in addition to the list of prior states that must be processed per state.[1] The overall increase in Harpy's speed is due to the small number of states (and consequently the small number of pointers and templates) that is checked per time sample, as shown in the above table.

4.8 System Tuning

The Harpy system is almost completely free of programmed speech-dependent knowledge; the only exception is the acoustic-template match routine. The speech knowledge of the system is contained in the templates, BNF syntax (via the network), lexical spellings and word juncture rules (again via the network), and phone durations (see section 4.4). Any errors that the Harpy system makes, assuming that the heuristic thresholds are properly set, must be attributed to one or more of these knowledge sources.

4.8.1 Lexical Generation

The first Harpy system used the original Dragon lexicon (see Baker (1975)). This lexicon is a dictionary of phonetic spellings and does not allow for the steady-state nature of the Dragon (and Harpy) system. For example, every word begins with a silence that is not optional, burst phones (such as [K], and [T]) do not allow for a silence part and an aspirated part and there is no accounting for voice bars caused by stops. Dragon is able to cope with many of these errors by its ability to over-parse and under-parse (see 4.2). However, these errors are magnified in the Harpy system because of segmentation and have a severe impact on the performance of the system. Harpy attempts to correct these errors by the use of a phonetic dictionary and word juncture rules that account for unique acoustic sounds in the pronunciations of the words.

[1]The "number of pointers" refers to the number of inter-state connections.

There is an obvious close relationship between the phonetic spellings of the lexical words and the acoustic-phonetic templates used. This gives rise to the "chicken-egg" problem of generation of dictionary and templates. Baker, in his Dragon system, started with a "standard phonetic" dictionary and generated as many templates as needed to match the acoustic signal of his training data to the lexicon. The dictionary was changed only when gross errors were detected (e.g., the use of [WH] instead of [K W] in the word "queen"). One could just as well start with a set of templates and generate the lexicon to match to the training data. The success of either scheme depends on the accuracy of the starting set (either dictionary of templates). The single major problem with Dragon's scheme is that it allows multiple templates per phone in order to match the dictionary to the training data. The result is that there are many overlaps among the phones within the parametric space (there were 168 templates covering 33 phones). For example, the silence phone ([-]) has several templates, some of which are the voice bars produced by the sonorant stops. These voice bar templates match very closely to nasals. The result is that silences in the lexicon match both to true silences and to nasals.

The "QTRAIN" experiment was the first attempt to produce an acoustic dictionary by starting with a set of templates.[1] This experiment was done using the ZCC parameters. The training data (same as used by Baker for the Dragon system) was tediously hand segmented and labeled. Goldberg (1975)[2] used this data to produce a set of templates that covered the acoustic space to within the limit set by Baker.[3] All templates produced were uniquely named and only after the lexicon was produced were some of the templates given similar names (there were 83 templates covering 62 phones).

The lexicon was produced in a semi-automatic fashion by considering together all occurrences of each word. The best seven template matches for each segment were manually examined to produce a dictionary spelling of each word (and possibly juncture rules) that best explained the acoustic phenomena. The resulting dictionary is shown in appendix A; the corresponding performance results are given in section 4.7. The results show that it is possible to start with a set of templates and produce a lexicon tuned to training data that gives good results.

There are many problems with the QTRAIN scheme: 1) The dictionary is tuned for the Dragon system rather than for Harpy. The result is that there are many "inclusive or" (see 4.3) choices in the dictionary which are needed to account for the oscillating nature of the 10ms samples among several templates. This becomes detrimental when the acoustic signal is segmented into larger sized time units, as in the

[1]This tuning was actually done for the Dragon system rather than the Harpy system

[2]My thanks to Dr. Goldberg for producing these templates for me.

[3]Each 10ms sample matched at least one of its corresponding templates to within a probability of at least .1.

Harpy system. 2) The ZCC parameters are unstable (this is the cause of condition 1 above). There is a great variation in the parameters, even for two separate occurrences of the same acoustic-phonetic sound within the same context. This variation is due mainly to changes in amplitude. 3) There is nothing "sacred" about the chosen set of templates. Each template center was chosen from one 10ms sample that best fit a large number of similarly labeled samples. A change in the tolerance limit for matching will produce a different set of templates. 4) Since the templates are not fixed, neither is the dictionary. A different set of templates will result in a different dictionary. 5) Concluding from the above, this scheme is not extendable to other speakers; a new set of templates and a new dictionary would have to be generated for a new speaker.

To make a system that is extendable to many speakers, a "standard" phonetic dictionary is needed that is composed of phones that represent speaker-independent acoustic sounds. The data used to accomplish this consist of two sets, each of the same 20 utterances spoken by four different speakers. One set from each speaker was used as training data and the second set was used as test data.[1] The Desk Calc task domain was chosen because of the large number of acoustically confusable words that it contains.[2] The templates were generated by "averaging" all occurrences of each phone in the training data (see section 4.9). A separate set of templates was generated for each speaker but a single phonetic dictionary was produced for all four speakers.

At this point the ZCC parametric scheme was abandoned in favor of the more accurate Itakura[3] metric. The lexicon was generated by starting with an "obvious" phonetic transcription of each word and then modifying it by the acoustic evidence in the training data to produce the final phonetic dictionary. Every word of every utterance in the training data was manually examined to confirm or alter the dictionary entry and to add to or modify the word juncture rules. An important point is that the templates were manually generated simultaneously with the dictionary generation and modification (see section 4.9).

After the initial dictionary and template sets were generated, the system was run on the training data with no syntactic support. This was achieved by generating a network from the dictionary, word juncture rules, and a BNF grammar that allows any

[1]The errors that the system makes on the training data are manually analyzed and the appropriate corrections are made. The system is repeatedly run on the training data to find and correct all errors that it makes. It is assumed, therefore, that the system will achieve 100% accuracy on the training data.

[2]The branching factor for this language is about 10.8. The confusable words "alpha", "beta", "gamma", and "delta" can all occur in syntactically equivalent positions. See Goodman (1976).

[3]See Itakura (1975).

word to follow any other word. The lack of syntactic support (and the resulting high branching factor of 37 at each word) allows acoustic ambiguities and errors to be spotted immediately by manual examination of the recognition results on the training data. The resulting errors were analyzed and attributed to either mis-tuned templates, lexical spelling errors, word juncture rule errors, or acoustic duration errors (for the data-dependent transition probabilities). The errors were corrected and the system re-run until the system achieved 100% (or near 100%) accuracy on the training data without syntactic support. The final dictionary produced by this tuning is shown in appendix B. The performance results with this tuning are given in section 4.9.

4.8.2 Template Generation

The lessons learned from the QTRAIN experiment about template generation are: 1) It is nearly impossible to find a single exemplary sample that can serve as the cluster center for any particular sound and 2) Errors in hand labeled data result in extraneous and errorfull templates being produced by an automatic template generation system. The templates generated for the Harpy system, in contrast, are manually selected and represent an averaging of many samples of the acoustic data for each phone. There is only one template per phone. Averaging for the Itakura metric is accomplished by summing the autocorrelation coefficients from the desired samples. This average cluster center, while it may not match perfectly to any single sample, is the only template which receives the highest (or near highest) probability match across all of its acoustic samples. This averaging is especially desirable in the Harpy system because of the averaging done by its segmenter (see section 4.6). Mistuned templates (templates that do not match "well" to an occurrence of its acoustic signal) are corrected by adding the autocorrelation coefficients of the acoustic samples that are in error to the cumulative sum of the cluster center for the template in question.

Intra-word allophonic variation is accounted for by the use of unique phone (and template) names within the dictionary. For example, there are six different variations for the phone "AH" (see appendix E). Each phone is uniquely named (by numbering) and each has one and only one template. There is some "art" needed in generating the templates and lexicon. Very often while generating the templates, a particular template that has been partially tuned will match very poorly to the acoustic signal of a word in which it should belong. One is tempted at this point to alter the lexical entry to account for this phenomenon. However, after averaging the template with the acoustic signal, it is usually found that the resulting template matches surprisingly well with both the current acoustic signal and all prior occurrences of this phone. The lexical entries are changed only as a last resort in tuning. This results in a "clean" looking dictionary (see appendix B) that can be easily used to tune templates for a new speaker to the system.

words within the utterance are correctly recognized), and TIME is the number of times real-time needed for recognition. TIME includes 8.4 times real-time for the generation of the autocorrelation coefficients and the linear predictor coefficients.

Complete details of the above results are given in appendix F. The dictionary used for the above test results is shown in appendix B, the word juncture rules are shown in appendix D, and the phone durations are shown in appendix E.

4.10 Program Space and Time

The Harpy system currently runs on a PDP KA-10 computer which is a 36 bit word machine, and has an effective speed of about 342,000 instructions per second. The space used by the program may be considered in two parts, a fixed amount for code and general data, and variable amount for data, depending on the network size, number of templates, etc. The fixed amount is 54K words and is composed of the following:

3K	output routines
1K	segmentation routine
1K	network searching algorithm
10K	autocorrelation coefficients generation routine
2K	initialization routines
1K	linear predictor generation routine
18K	general global data
5K	debugging routines
6K	library routines (LOG function, I/O, etc.)
7K	general storage (stacks, string space, etc.)
54K	

The breakdown for the variable data storage (in words) is as follows:

9 x number of network states
18 x number of 10ms samples used per utterance
(currently 600 = 6 seconds maximum)
5 x number of pointers (i.e., inter-state transitions, approximately)
19 x number of templates
4 x number of phones
(usually the same as # of templates)

The variable data sizes (in K words) for the Desk Calc task with syntax are:

4.8.3 Word Juncture Rule Generation

The word juncture rules are used to describe acoustic-phonetic phenomena that occur at word boundaries. Each juncture rule contains a left and right context of a word boundary and a description of what changes may occur at this juncture. A complete description of the word juncture rules is given in 5.2.3. The word juncture rules are generated simultaneously with the dictionary and the templates. Tuning of the juncture rules consists of either adding new rules or altering existing ones to account for new occurrences of juncture phenomena.

4.9 Performance

After the system was tuned on its training data, it was run on its test data. The test data consist of two large sets, each of which contains four smaller sets, one from each of four speakers. All three sets spoken by each of the four speakers (one training set and the two test sets) contain the same 20 sentences and 110 words. The first test set was recorded during the same session as the training set; the second test set was recorded five months after the training set. No analysis has been done on any of the errors made by the system on the test data, nor has any attempt been made to correct these errors. Error corrections were done only on the training set. Each test set was run in two modes: with syntactic support and without syntactic support. The network for the latter mode was created by using a BNF grammar that allowed any word to follow any other word. The effective syntactic branching factor (the average number of words that can follow any other word in a sentence) for the runs with syntax is about 11, and without syntax it is 37. See Goodman (1976) for a complete treatise on the complexities of these two grammars. The following are the overall results of the Harpy system on the two sets of test data:

TEST DATA	%WORDS	%UTTS	TIME
1) WITH SYNTAX	97.5	88.6	13.1
1) NO SYNTAX	90.8	55.0	18.0
2) WITH SYNTAX	93.1	70.9	13.2
2) NO SYNTAX	83.5	36.7	18.0

"1)" and "2)" indicate the immediate and five month test sets respectively. %WORDS is % words correctly recognized, %UTTS is the % utterances correctly identified (i.e., all

7.4K	network states
10.8	speech data
17.5	network pointers
1.1	templates
.3	phones
37.1K	

The variable data sizes (in K words) for the Desk Calc task without syntax are:

2.2K	network states
10.8	speech data
11.4	network pointers
1.1	templates
.3	phones
25.8K	

The total size of Harpy for the Desk Calc task is about 91K with syntax and about 80K without syntax.

A breakdown of the time needed for the recognition of an average utterance that takes 13.1 times real-time is as follows:

%TIME	# TIMES REAL-TIME	ROUTINE
57.5	7.53	autocorrelation coefficient generation
6.6	.87	linear predictor coefficient generation
12.0	1.57	segmentation
12.5	1.64	state probability updating
.3	.04	backtracking
11.1	1.45	acoustic-phonetic template matching
	13.1	

The above tables indicate that the majority of program space and time is taken in the generation of the autocorrelation coefficients and the linear predictor coefficients. The search algorithm itself is very small (2% of the fixed program space and about 1.5% of the total space). The major portion of the data space used is for storing the network data. This may be the major problem with systems of this type for larger grammars (see section 6.3).

5 NETWORK DATA REPRESENTATION

5.1 Description and Format

The networks as used by the Harpy and Dragon systems are a tractable method for representing the combined syntactic, lexical, and, for Harpy, word juncture knowledge. A network is a set of states with inter-state connections and, in the case of Dragon, inter-state and intra-state transition probabilities. The networks for Harpy do not contain transition probabilities (See section 4.4). The networks are generated by a network compiler from the BNF grammar, the phonetic dictionary, and the word juncture rules.[1] Each state in the network contains the following information: the word (the terminal symbol from the BNF grammar), a unique ID number (every BNF terminal symbol is given a unique number), the phone (from either the phonetic dictionary or the word juncture rules), a list of prior states (all the states that may transition into the current state), and, for Harpy, a list of following states (all the states to which the current state may transition). The words that represent the recognized utterance are taken from the states that lie along the global path. It is important to note that each word of the recognition usually occurs in more than one state. For example, the word "one" may occur in at least three states, namely the states that represent the phones [W], [AH], and [N]. If a word (i.e., terminal symbol of the BNF grammar) can legally occur several times in a row, then the recognition system must have a means of detecting that the multiple states (on the global path) which contain the same word (terminal BNF symbol) actually represent more than one occurrence of the symbol in the BNF grammar. The unique ID number is used to indicate to the recognition system when the global path actually passes through a different BNF terminal symbol.[2] There are two unique states in the network: the initial state, which has no prior states, and the final state, which has no following states. These two states are indicated in the

[1]A network should not be confused with the derivation tree of a context free language (a BNF grammar) (see Hopcroft, et al. (1969)). The nodes of a derivation tree that are not terminal represent the non-terminal symbols of the grammar while every node (called state) of a network represents a terminal symbol of the grammar. Also, a derivation tree does not allow loops while a network does.

[2]This causes a small problem in the BNF grammar specification. A terminal symbol in the BNF must never be allowed to immediately follow itself, otherwise the recognition system would not realize that there is a loop in the global path which passes through a terminal symbol more than once. In these cases, the BNF must specify two alternating terminal symbols of the same name. See one of the "DIGIT'"s specification in the desk calc BNF grammar in appendix C.

BNF by the symbols "[" and "]", respectively. Every path[3] that goes from the initial state to the final state represents a syntactically and phonetically legal path. Also, the set of all such network paths is exactly the set of all legal syntactic-acoustic paths.[1]

5.2 Generation

Baker (1975) gives the basic method for generating a network from the BNF grammar and a phonetic dictionary. The networks are generated in several passes: the BNF grammar is used to generate a grammar network; each state in the grammar network, which represents a terminal symbol in the BNF grammar, is expanded with a phonetic subnetwork that represents the dictionary spelling for the terminal symbol; word juncture rules are added; and, finally, special network reduction heuristics are applied.

5.2.1 BNF Grammars

The syntactic knowledge contained within the networks comes from the BNF grammar specification (See appendix C for an example). The network compiler reads the BNF specification and generates a grammar network in which every state represents a terminal symbol of the BNF. A path through the network is a contiguous sequence of states that are connected by inter-state connections. Each path through the network from the initial state to the final state represents an utterance (sentence) of the language that is specified by the BNF grammar. A BNF grammar is not the most efficient means to specify a network because of the possibility of either NIA's (see below) or redundant states (see section 5.3.2). However, a BNF is a convenient means of defining an artificial language for a specific task domain in a connected speech recognition system because the BNF represents the productions rules of a context free grammar and the parsing of a context free grammar is well understood.

One major problem that may be encountered with using a BNF grammar specification to generate a network is what Baker refers to as a loss of context for a

[3]A "path" is a sequence of states that is connected by inter-state connections (or pointers).

[1]This may not be true in the case of Non-terminal Intersection Ambiguities (explained below).

non-terminal symbol when it is used in more than one unique context. We call this "Non-terminal Intersection Ambiguity" (NIA). NIA's are not inherent in the BNF grammar specification, but occur because of the way in which the BNF is used to generate the network. This can best be explained by an example. Consider the two partial BNF productions:

(left-context-1) <non-terminal-x> (right-context-1)
(left-context-2) <non-terminal-x> (right-context-2)

The <non-terminal> symbol has been used in more than one unique context (both left and right). The production for <non-terminal-x> (subnetwork) will be expanded once and all references to this non-terminal (occurrences of it on the right hand side of the productions) will create links to and from this subnetwork. The resulting network will contain not only the legal paths "(left-context-1) <non-terminal-x> (right-context-1)" and "(left-context-2) <non-terminal-x> (right-context-2)" but also the illegal paths "(left-context-1) <non-terminal-x> (right-context-2)" and "(left context-2) <non-terminal-x> (right-context-1)". There is no known practical automatic way of detecting and eliminating the NIA's from a grammar because similar contexts for a non-terminal symbol cannot be accurately determined during the network generation.[1] With one notable exception, all the BNF grammars that are used by the Dragon and Harpy systems were hand edited to remove NIA's. The exception is the formant task grammar (see Baker, 1975). In this grammar, the non-terminal symbol <f!nine-digit> occurs in at least 12 unique contexts. This is compounded by the fact that within the <f!nine-digit> production, the non-terminal symbol <f!three-digit> occurs in three different contexts. Further, the non-terminal symbol <f!digit> occurs in two different contexts within the <f!three-digit> production. The result is that there would have to be at least 72 (2x3x12) different productions for <f!digit> to resolve the NIA's for the <f!nine-digit> non-terminal symbol only. There are other NIA's in this grammar. The point of all of this is that there are some task networks which contain illegal paths because it is impractical, if not impossible, to remove all NIA's. The correct recognition path must be resolved by the lower level knowledge sources in these cases. This is strong motivation for accurate lexical and acoustic knowledge within the system.

[1]One method for eliminating NIA's from a grammar network is to generate a separate unique production (subnetwork) for every occurrence of a non-terminal symbol on the right hand side of a production. However, this becomes impractical for large grammars because of the large number of productions generated and, in fact, impossible for recursive grammars.

5.2.2 Dictionaries

Each state of the grammar network (which represents a terminal symbol) is expanded with a subnetwork, that represents the pronunciation(s) of the terminal symbol word, to form the complete syntactic-phonetic state network. These subnetworks are generated from a phonetic dictionary (see appendices A and B).

The phonetic dictionary that Dragon uses contains only one phonetic transcription per word entry; alternate transcriptions are specified by multiple word entries. This becomes unwieldy for words that have several choices for more than one phone in the word because of the multiplicative explosion (see below for an example). Therefore, the Harpy dictionaries have only one phonetic transcription entry per word with alternate choices being indicated by a special notation. This notation consists of parentheses to indicate alternative choices, and commas to separate the alternatives. The special character "0" is used to indicate the null choice. Parentheses may be nested in any manner (provided they are paired). For example, the entry for the word "into":

```
(-,0)((IH,IX,0) N - T,IX N DX) AX
```

is equivalent to the eight entries:

```
- IH N - T AX
- IX N - T AX
- N - T AX
- IX N DX AX
IH N - T AX
IX N - T AX
N - T AX
IX N DX AX
```

Further, the square bracket pair "[]" is used to indicate optional acoustic choices within a single state. That is, the list if phones within the "[]" pair generates one state only and the entire list of phones will be assigned to that state as optional choices. Nesting of the the "[]" is not allowed. For example, the spelling for "four", [K1,F] AO2 [ER1,ER4], generates three states; the first state allows for the occurrence of either the phone [K1] or [F], the second allows only the phone [AO2], and the third allows for the occurrence of either the phones [ER1] or [ER4]. Logically, the "[]" pair encloses <u>inclusive or</u> choices of phones whereas the "()" pair encloses <u>exclusive or</u> choices of phones. (see QTRAIN dictionary in appendix A).

5.2.3 Word Juncture Rules

The word juncture rules are added to the network, after the acoustic spellings have been incorporated, in order to resolve possible inter-word juncture phenomena. Appendix D shows the word juncture rules. The syntax for the juncture rules is as follows:

```
<rule>::=             <left-context>,<right-context><opcode><phonetic-trans>
<left-context>::=     <phone-choice>
<right-context>::=    <phone-choice>
<phone-choice>::=     <j-phone>|[<phone-list>]
<phone-list>::=       <j-phone>|<j-phone>,<phone-list>
<j-phone>::=          <any-legal-phone>|<|>|0|#
<opcode>::=           ↑|;|{|}|←|→
```

Juncture rules are applied only at word junctures[1] within the network during generation. The <left-context> and <right-context> are the phone choices for the last phone of the left word and the first phone of the right word, respectively, at the juncture. There are four special symbols that can be used in the context specification. They are: "<" and ">" for left phone and right phone respectively, "0" for any phone except for the initial and final states, and "#" for the initial state if used in left context or final state if used in right context (i.e., the start or end of utterance, respectively). For example, the rule:

```
0,<{
```

is applied at junctures where there is anything on the left (except the initial state) and the right phone matches the left. The following rule would have the same effect:

```
>,0{
```

[1]Intra-word acoustic juncture phenomena are accounted for in the dictionary.

The semantics for the opcodes are:

↑ substitute the <phonetic-trans> in place of both phones
; insert the <phonetic-trans> between the two phones
{ make the left phone optional
} make the right phone optional
← remove the left phone (and link around it)
→ remove the right phone (and link around it)

The <phonetic-trans> is a dictionary-like entry with the following exceptions: the "<" and ">" characters may be used to indicate the left and right phones respectively; all phones given within the <phonetic-trans> are assumed to be associated with the right hand lexical word; and a "." is used in front of a phone to force it to be associated with the left lexical word. For example, the following rules are equivalent:

```
>,0{
0,<↑((.<,0) >)
```

Another example, the rule "ER,#↑(.<,.R)(.AXX,0)" when applied to the word "four" ("(-,0) F AH4 ER") at the end of the sentence has the effect of altering the phonetic spelling to "(-,0) F AH4 (ER,R) (AXX,0)".

The word juncture rules are included with the phonetic dictionary to indicate the complete options of the word pronunciations in all contexts. The generation of the word juncture rules is part of the overall system tuning (including the dictionary generation and template generation).

5.3 State Size Reduction

From the Dragon time model (see section 3.6), one sees that, except for a constant overhead, the recognition time for a given utterance is nearly linearly dependent on the network size. As was shown with Harpy (see section 4.7), the recognition time is dependent upon the confusibility among the paths. Therefore, several techniques were developed to reduce the network size and complexity (the number of confusable paths). They are removal of null states, removal of redundant states, and subsumption of common states.

5.3.1 Null State Removal

A null state is a state which contains no pertinent syntactic or phonetic information. That is, a null state contains no information other than the connections to its prior and following states; it is a "dummy" state. The null states as they appear in Dragon's networks, are the "ghosts" of the non-terminal symbols of the BNF grammar. It is an expedient technique to utilize null states during the network generation because they are the common juncture links between many incoming and outgoing states. These null states have little or no effect on the Dragon system because the system is able to "hide" them in the many 10ms time samples during the recognition process by underparsing them,[1] i.e., by absorbing them into the many 10 milli-second time samples. However, it becomes essential to remove the null states for the Harpy system because this system operates on larger time units, due to segmentation, than the 10ms time level (see section 4.5). A segment is too large a time unit to be "thrown away" in "hiding" a null state (i.e., too much acoustic information is lost). A null state is removed by linking each of its prior states to each of its following states and deleting the null state.[2] This results in a reduction in the total number of states of between 7% and 26%, depending on the BNF grammar. However, the efficiency of this state reduction is tempered by the fact that there is an increase in the number of inter-state links ("pointers"). If a null state has N prior states and M following states, its removal reduces the state size by one but increases the number of pointers by MxN-(M+N).

5.3.2 Redundant State Removal

A considerable savings in both state and pointer size can be realized through the technique of removing redundant states. This technique is a heuristic (as opposed to an algorithm) and does not guarantee a minimal network.[3] However, it is a simple heuristic to implement that is guaranteed not to alter the semantic content of the network (i.e., the grammar specification). Redundant states can occur in both the grammar network and the complete network.

[1] The intra-state transition probability for these null states is set to 0.

[2] "Delete" implies removing all links to and from the state being deleted.

[3] See Hopcroft and Ullman (1969) and Karp, et al. (1974)

Two states A and B in the grammar network are redundant if: 1) each has have the same terminal lexical symbol (but different ID numbers) and each has exactly the same prior states, or 2) each has the same terminal lexical symbol and exactly the same following states.[1] When two states, assume A and B, are found to be redundant, one of them can be removed from the network without any loss of syntactic information in the following way: One of the states is selected for removal (they are equivalent). Assume, for the sake of argument, that the two states satisfy condition 1 and that state A is to be removed. Removal is then accomplished by linking state B to all the following states of A (if they are not already linked) and deleting state A from the network. Similarly if the two states satisfy condition 2, then all of the prior states of state A are linked to state B (again if they are not already linked) and state A is deleted. It is important to note that by removing one redundant state from the network others may be created by the relinking process. Therefore, the network must be repeatedly searched until no more redundant states are found.

For example, consider the following BNF grammar:

```
<sentence>::=   [ <request> ]

<request>::=    TELL <pron> <quan> THE <sum>
                TELL <pron> ABOUT <det>
                GIVE ME THE <sum>

<pron>::=       US
                ME

<quan>::=       ABOUT
                ABOUT ALL

<sum>::=        STORIES
                HEADLINES

<det>::=        NIXON
                CHINA
```

Figure 5.1 shows the grammar network that is generated from this BNF (with null states removed). There is a state in the network for every occurrence of a terminal symbol in the BNF. The two "TELL" states are redundant by condition 1, as are the three "ABOUT" states. The two "THE" states are redundant by condition 2. This gives a total of four redundant states and nine pointers that may be removed. Figure 5.2 shows the resulting network after the redundant states are removed. The two sets of syntactic paths specified by the two networks are identical; there has been no loss of syntactic information.

[1]"Same prior" and "same following" states mean <u>identically</u> the same states, not just the same terminal lexical symbol. This constraint guarantees the integrity of the semantic content of the network.

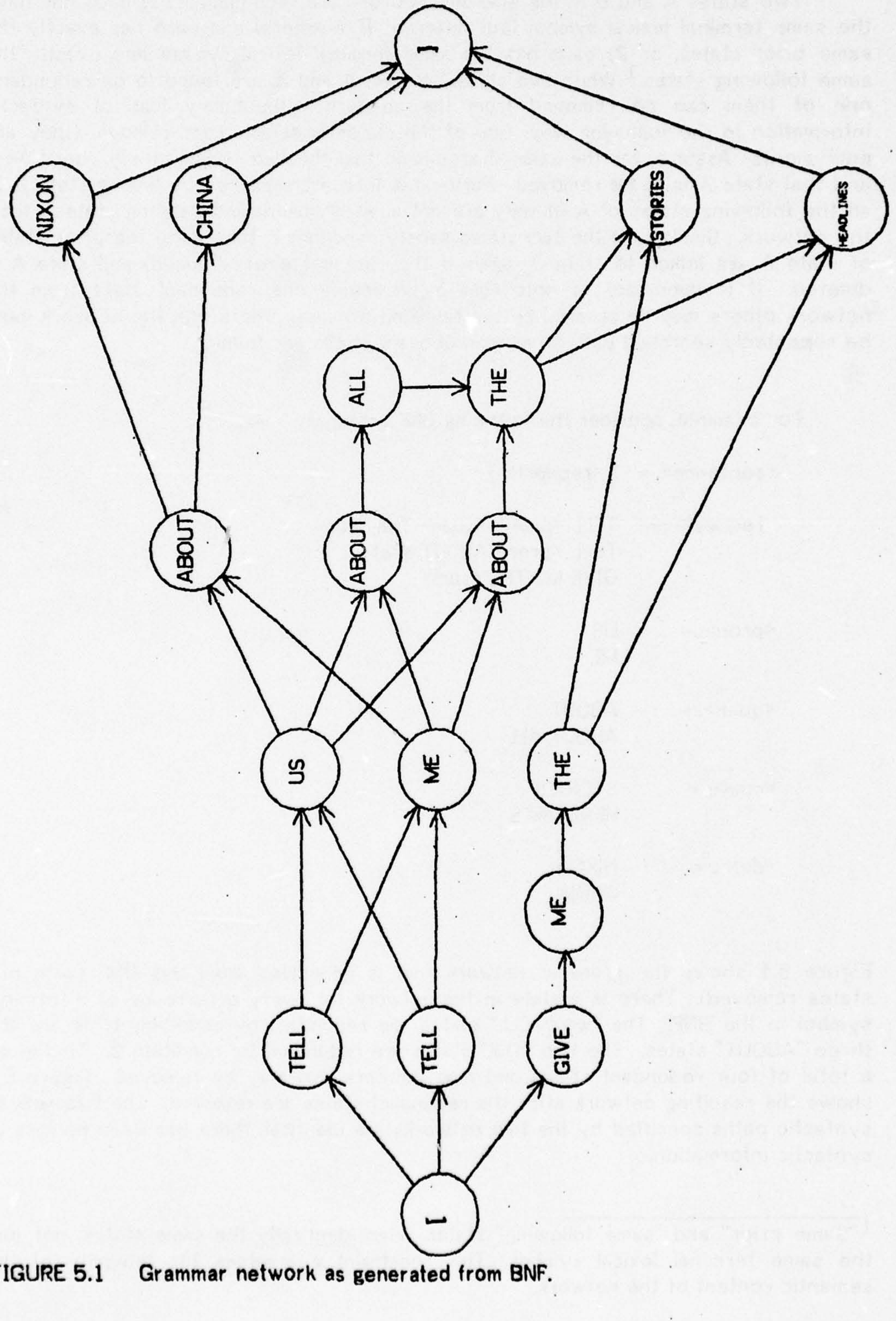


FIGURE 5.1 Grammar network as generated from BNF.

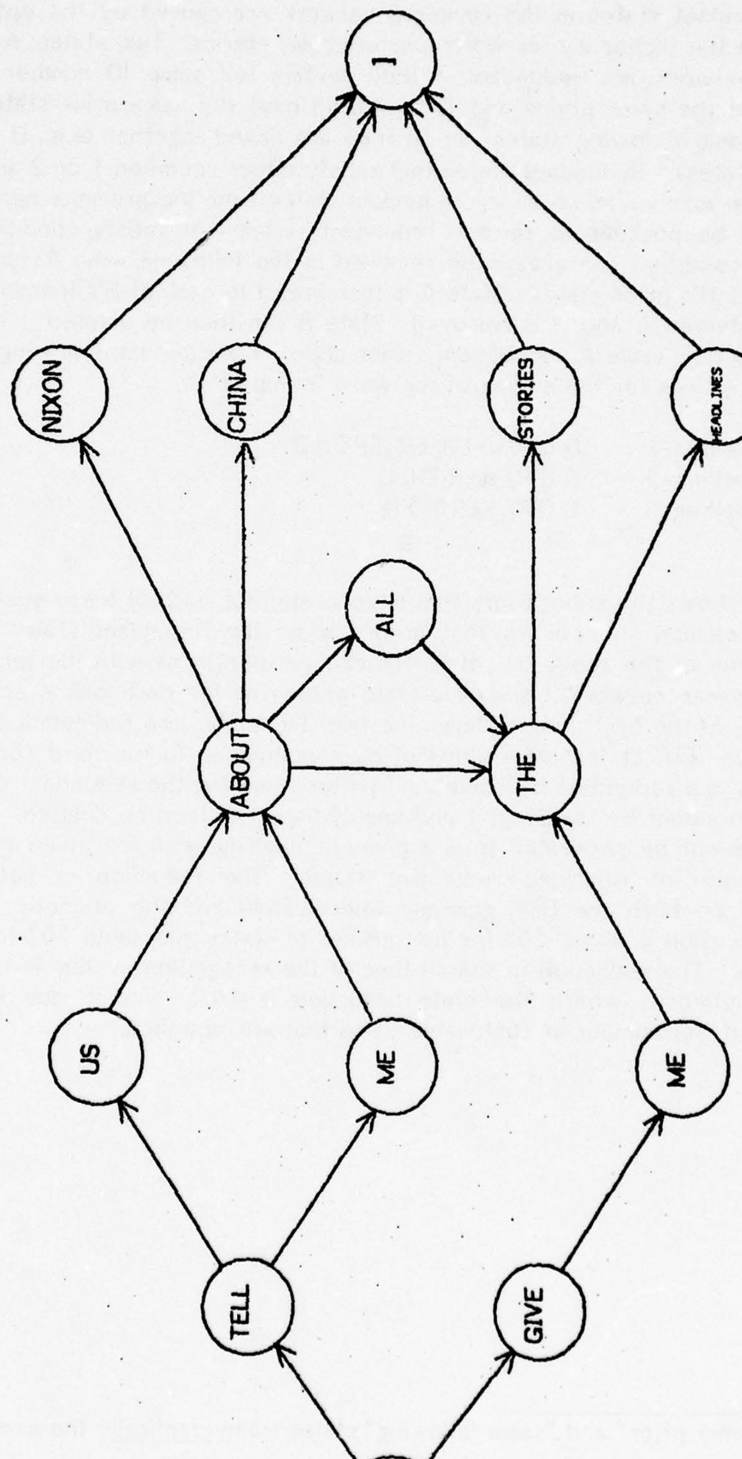


FIGURE 5.2 Grammar network with redundents removed.

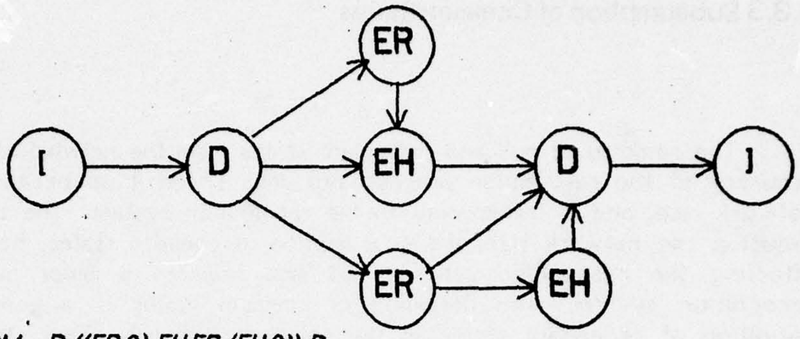
Redundant states in the complete network are caused by the optional choices specified in the dictionary (or word juncture rule) entries. Two states, A and B, in the complete network are redundant if they contain the same ID number (and lexicon symbol) and the same phone and 1) they both have the same prior states or 2) they have the same following states[1] or 3) they are linked together (e.g., B is one of A's following states).[2] Redundant states that satisfy either condition 1 or 2 are removed in the identical manner as removing redundant states from the grammar network. It may not always be possible to remove redundant states that satisfy condition 3 but the redundant condition can always be removed in the following way: Assume state A is one of state B's prior states. State A is then linked to each of B's following states and the link between A and B is removed. State B can then be deleted if it now has no prior states (i.e., state A was B's only prior state). Consider the following three partial phonetic spellings for the ending of the word "hundred":

spelling-1	D ((ER,0) EH, ER (EH,0)) D
spelling-2	D (ER,EH,ER EH) D
spelling-3	D (ER,0)(ER,EH) D

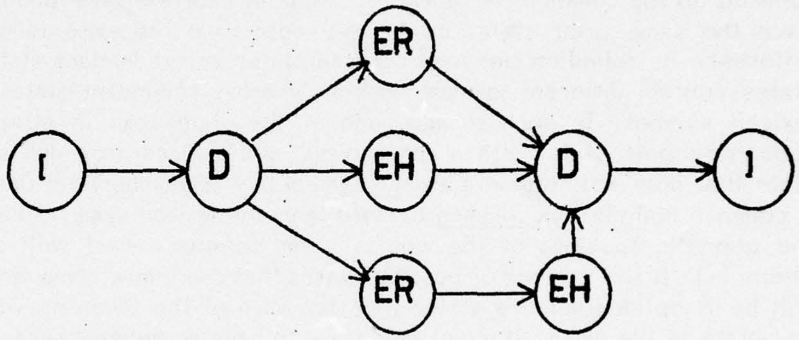
Figure 5.3 shows the subnetwork that is generated for each of these spellings plus the resulting common subnetwork that occurs when the redundant states are removed from any one of the above (i.e., they are all equivalent). As with the terminal symbols in the grammar network, there is a state generated for each phone specified in the dictionary. In the first two spellings, the two "ER" states are redundant by condition 1 and the two "EH" states are redundant by condition 2. In the third spelling, the two "ER" states are redundant by condition 3; after removing the redundant condition, they become redundant by condition 1 and one of them can then be deleted. The collapsed network cannot be generated from a phonetic spelling, with the given syntax, without the technique of removing redundant states. The reduction in network size is dependent on both the BNF grammar specification and the phonetic dictionary. A typical reduction is about 10% for the number of states and about 20% for the number of pointers. The reduction in search time of the recognition system is typically about 30% for networks where the state reduction is 10%. This is due mainly to the reduction of the number of confusable paths that are searched.

[1]Again, "same prior" and "same following" states mean <u>identically</u> the same states.

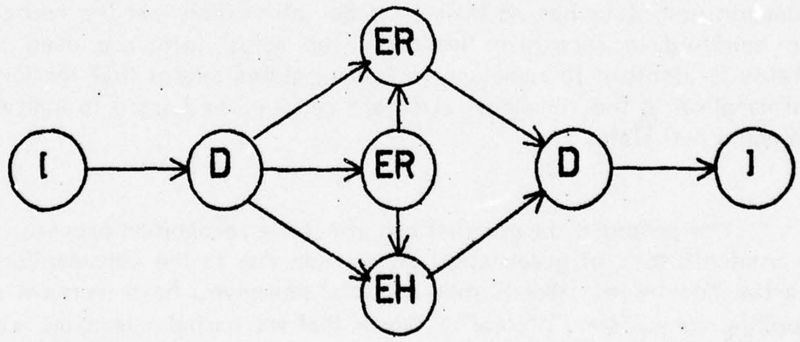
[2]This is a redundant condition only because there is an implied intra-state link for all states in the complete network.



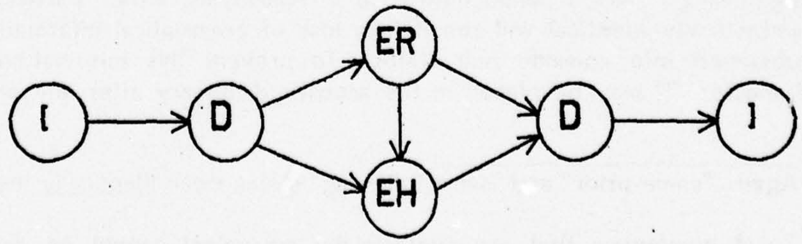
NETWORK FROM D ((ER,0) EH,ER (EH,0)) D



NETWORK FROM D (ER,EH,ER EH) D



NETWORK FROM D (ER,0) (ER,EH) D



ANY OF THE ABOVE NETWORKS AFTER REMOVAL OF REDUNDANT NODES

FIGURE 5.3 Examples of phonetic networks.

5.3.3 Subsumption of Common States

The removal of null and redundant states from the network does not affect the accuracy of the recognition process, but does speed it up because of the reduced network size, and is transparent to the recognition system. The third technique for reducing the network size, the subsumption of common states, has the potential of affecting the recognition process and also requires a minor modification to the recognition system. The definition of common states is a generalization of the definition of redundant states in the complete network. Two states A and B are common (in the complete network) if: they both have the same phone and 1) they both have the same prior states or 2) they both have the same following states.[1] The difference in definition between common states and redundant states is that common states contain different lexicon symbols whereas redundant states contain the same lexicon symbol. In such a case, one of the states can in effect be removed by subsuming both states into a common null state. A common null state is a "dummy" state that does not contain a lexicon symbol (nor ID number) but does contain a phone. A common null state is <u>common</u> to more than one lexicon symbol. For example, most of the phonetic spellings of the words in the dictionary start with an optional silence phone [-]. If the number of possible words that can follow some word is N, then there will be N optional silence states (one for each of the following words) following the last state of the word. It is not necessary to have an optional silence state for each of the following words; one common null state (of silence) will suffice. It is because the common null state has no lexical symbol information that the recognition system must be modified to recognize this fact. The actual technique used to remove common states is identical to removing redundant states except that the lexical symbol and ID information in the remaining state are removed or tagged to indicate that it is now a common null state.

The potential danger that can affect the recognition process is that there can be a complete loss of grammatical information due to the subsumption of either total or partial homonyms. Words that are total homonyms have identical phonetic dictionary spellings (e.g., "steel", "steal").[2] Words that are partial homonyms are ones in which the dictionary spelling for one word is completely contained in the other word's spelling (e.g., "king", "king's" and "onto", "to"). Homonyms, either partial or total, that are syntactically identical will result in a loss of grammatical information when they are subsumed into common null states. To prevent this information loss, the special character "!" may be placed in the acoustic dictionary after any phone to indicate to

[1]Again, "same prior" and "same following" states mean <u>identically</u> the same states.

[2]Total homonyms that are syntactically equivalent cannot be distinguished by the recognition process and one or the other would be randomly chosen. However, if the total homonyms are subsumed into common null states then the recognition process could no longer make the random choice because all syntactic information would be lost.

the network compiler that the state that is generated for this phone is not to be subsumed into a common null state. It is to be considered an uncommon state. The words "IN" and "INTO" in the Desk-calc task dictionary contain these uncommon state markers (see appendix C).

The savings realized by the subsumption of common states is typically about 30% for the number of states and about 50% for the number of pointers. The savings is more significant for the Harpy system than the Dragon system because the number of confusable paths is considerably reduced by the collapsing of many acoustically identically paths into one. The effect on Harpy is a better than two fold speed up.

5.3.4 Intersection States

Recent experiments with large vocabulary grammars of about 1000 words (see appendix H) and large branching factors of about 30, have necessitated the development of another technique for the reduction of the number of pointers needed for the network representation. Recall from 5.3.1 that by removing a null state, the number of pointers needed increases. In the large branching factor grammars where the number of forward and back pointers of a null state can be 30 or more, then the cost[1] of removing them becomes prohibitive. To save the exponential blow up in the number of pointers needed, the offending null states are re-defined to be "intersection" states. An intersection state is a dummy state that is the common intersection between several incoming and outgoing states. An intersection state does not contain the usual state information such as lexical word, phoneme, unique ID number, etc., and the probably of being in an intersection state during the recognition process is (by definition) 0. States which link to an intersection state are logically linked to the states that the intersection state are linked to, rather than to the intersection state itself. The intersection state is considered to be an indirect list of pointers during the recognition process. This requires a modification to the recognition system to recognize the intersection states and act accordingly.

[1]To remove a null state with 30 forward and back pointers requires 840 more pointers than it takes to leave it in ((30x30) - (30+30)).

6 CONCLUSIONS

6.1 Introduction

The performance of the Hearsay-I system shows the power that speech-dependent heuristics have in improving the search speed by reducing the search space. Dragon shows the feasibility of searching through a network that contains the integral knowledge of both syntax and phonetics for the optimal acoustic path. Harpy combines the network representation of knowledge with heuristic techniques for reducing search space to produce a high speed and high accuracy system. The networks of the Harpy system differ from the Dragon networks in that Harpy uses connection networks while Dragon uses Markov networks. The following are some of the design choices of the Hearsay-I, Dragon, and Harpy systems that affect speed:

	Hearsay-I	Dragon	Harpy
Search strategy	Best-first with backtracking	All paths in parallel	Best few paths in parallel
segmentation	yes	no	yes
phonetic match	match all phones	match all phones	selective match

The design choices of the three systems that affect accuracy are the following:

	Hearsay-I	Dragon	Harpy
Word representation	base form dictionary	base form dictionary	network (machine derived)
juncture rules	procedural	none (can be added)	yes
prosodic rules	procedural	none	duration
parametric representation	5 octave band filters	5 octave band filters	LPC

Some interesting features of the Harpy system are the following:

1. The system was tuned on one set of data and tested on two separate sets of data for four different speakers. One phonetic dictionary is used for all speakers.

2. Harpy is faster than any other connected-speech system currently running. Harpy requires 4.5 million machine instructions per second of speech for recognition. Reddy (1976) gives the following rates for four other system (in million instructions per second of speech): Hearsay-I 3-15, Dragon 15-60, Lincoln 45-75, IBM 30.

3. Harpy is more accurate than any other system currently running. Harpy achieves 88.6% utterance accuracy for four speakers on a set of test data that was recorded with its training data and 70.9% utterance accuracy on a set of test data that was recorded five months after its training data. The accuracies reported for the above four systems are: Hearsay-I 32%, Dragon 48%, Lincoln 49%, IBM 55% (on two speakers). The branching factor (the average number of words that can follow any word in the syntax) of 10.8 for the task used in this study for Harpy is higher than for the other existing systems. The branching factor for the Lincoln task is 7.2 and for IBM 7.3.

6.2 Contributions

The contributions made by the Harpy connected speech recognition system are:

1. Harpy offers a viable compromise between a best-first search and a complete parallel search of all the syntactic paths. Harpy's technique of searching a variable number of "best" paths (both syntactic and phonetic) in parallel offers the same accuracy as searching all paths in parallel while taking only a fraction of the time (typically, about 7% of the time needed to search all paths in parallel). Harpy demonstrates that there is a simple heuristic which can be used to determine which are the "best" paths to follow; namely, only those states which are within a threshold amount of the highest state probability at every speech time sample need be followed.

2. Inter-state and intra-state transition probabilities, for this type of recognition system, will produce more accurate acoustic-phonetic mappings and, therefore, more accurate recognition results, if they are calculated dynamically during the recognition process from speech dependent-knowledge rather than a-priori from statistical measurements.

3. There are simple heuristics that can be used to reduce the network size, complexity, and confusibility among alternate paths. These heuristics are the removal of redundant states and subsumption of common states.

4. The accuracy of a speech recognition system is greatly improved by using speech evidence in producing the dictionary and templates. The dictionary of the Harpy system was produced by a careful study of the realizations of the words in the training data. Allophonic variations are specified in the Harpy system by the use of many unique phones (60) with one template per phone rather than a small number of phones with a large number of templates per phone.

6.3 Future Work

The system will need to be tested on a larger grammar than the 37 word desk calc grammar, preferably between 200 and 1000 words. The grammar should be carefully selected to reduce the a-priori acoustic ambiguities as much as possible.[1] New techniques for obtaining the lexicon and the user dependent-templates will have to developed as the current manual method will prove to be too tedious for 1000 word grammars. The time needed to produce the lexicon for the 37 word desk calc task was four hours for each of the four training sets, or a total of about 16 hours. Extrapolation implies that it will require more than 400 man-hours to produce the lexicon for a 1000 word task. The size and complexity of a network for a large grammar may require new methods for representation and storage; the network may be too large to fit in main memory of the current system. A rough estimate indicates that the network for a 1000 word task (assuming there are 22,000 states and 95,000 pointers) will require over 670K words of storage for representation of this network in the current system. While working with the Harpy system, the author observed that the speed of the system is roughly inversely proportional to the log of the network size. For example, consider the follow observed search times (excluding parameter calculations), in number of times real-time, and network state sizes (for the ZCC parametric scheme):

STATES	TIME
323	1.5
439	1.9
592	2.5
973	4.1
2250	5.8

The formula for recognition time is roughly

$$\text{TIME} = 4.4 \times \text{LOG}(\text{STATES}) - 9.7$$

If the extrapolation is accurate, then the search time (excluding the autocorrelation and linear predictor coefficient calculation time) should increase by only about a factor of two for a 1000 word grammar (assuming 22,000 network states) over the 37 word desk calc grammar (assuming 820 network states). However, there may be an overhead added for the storage scheme (possibly paging) of the larger network.

The majority of time used by the system (more that 64%) is for the generation of the autocorrelation and linear predictor coefficients. This problem is not unique to Harpy; all speech systems that use this parametric scheme must face this problem. The

[1] See Goodman (1976).

actual code used by Harpy for calculating these parameters has been streamlined by the removal of all loops and array subscripting. Instead of having an inner loop code that is called on 1500 times, Harpy replicates the inner loop code 1500 times and executes it linearly. Despite this effort at optimization, it still takes 8.4 times real-time to generate these parameters. It seems unlikely that an improved algorithm can be found to speed this calculation since the problem lies in the shear volume of data that must be used in calculating these parameters (the digitized wave form). A possible solution for improving the speed of this calculation may lie with hardware technology (e.g., a special high speed hardware device specifically designed to do these calculations). Another possibility is the use of another parametric scheme.

6.4 Summary

The Harpy system described in this thesis is an example of a fast and accurate connected speech recognition system which is also conceptually simple. The Harpy system, in its present state, appears to be faster and more accurate than any other speech recognition system that has been reported. This is not to imply that the design choices used in the Harpy system (e.g., data representation, search, and speed up heuristics) are either ideal or optimal techniques for recognition of connected speech but rather they represent one point in the evolution of speech recognition systems and help point the way for further research.

REFERENCES

Baker, J.K. (1975), "The DRAGON System -- An Overview", IEEE Trans. ASSP-23, 24-29.

Baker, J.K. (1975), "Stochastic Modeling as a Means of Automatic Speech Recognition", (Ph.D. Thesis, Carnegie-Mellon Univ.), Tech Rep., Comp.Sci. Dept, Carnegie-Mellon University.

Baker, J.K. and Bahl, L. (1975), "Some Experiments in Automatic Recognition of Continuous Speech", Proceedings Eleventh Annual IEEE Computer Society Conference, 326-329.

Erman, L.D. (1974), "An Environment and System for Machine Understanding of Connected Speech", (Ph.D. Thesis, Stanford Univ.), Tech. Rep., Comp. Sci. Dept., Carnegie-Mellon University.

Erman, L.D. (ed.) (1974), Contributed Papers of IEEE Symposium on Speech Recognition, Carnegie-Mellon Univ., (IEEE cat. NO. 74CH0878-9 AE).

Erman, L.D., Lowerre, B.T., and Reddy, D.R. (1973), "Representation and Use of Acoustic-Phonetic Knowledge in the HEARSAY System", 86th Mtg Program, Acous. Soc. Am., Los Angeles, 49 (abstract).

Forgie, J.W., Hall, D.E. and Wiesen, R.W. (1974), "An Overview of the Lincoln Laboratory Speech Recognition System", J. Acoustic Society of Amer., 56, S27 (A).

Goldberg, H.G. (1975), "Segmentation and Labeling of Speech: A Comparative Performance Evaluation", (Ph.D. Thesis, Carnegie-Mellon Univ.), Tech. Rep., Comp. Sci. Dept., Carnegie-Mellon University.

Goldberg, H.G., Reddy, D.R., and Suslick, R. (1974), "Parameter-Independent Segmentation and Labeling of Speech", in Erman (ed.).

Goodman, G. (1976), "Language Design for Man-Machine Communication", (Ph.D. Thesis), Comp. Sci. Dept., Carnegie-Mellon University, (in preparation).

Hopcroft, J. E. and Ullman, J. D. (1969), Formal Languages and their Relation to Automata, Addison-Wesley Publishing Co., Mass.

Itakura, F. (1975), "Minimum Prediction Residule Principle Applied to Speech Recognition", IEEE Trans. ASSP-23, 67-72.

Karp, R.M., Miller, R.E., and Rosenberg, A.L. (1974), "Rapid Identification of Repeated Patterns in Strings, Trees and Arrays", IBM T. J. Watson Research Center, Yorktown Heights, N.Y.

Lowerre, B.T. (1974), "A Camparison of Two Speech Understanding Systems", 88th Mtg Program, Acous. Soc. Am., St. Louis, 27 (abstract).

Markel, J. (1972), "The SIFT Algorithm for Fundamental Frequency Estimation", IEEE Trans. AU-20.

Markel, J.D. and Gray, A.H. Jr. (1973), "SCRL Linear Prediction Analysis/Synthesis Programs", SCRL, Inc., Santa Barbara, Calif.

Neely, R.B. (1973), "On the Use of Syntax and Semantics in a Speech Understanding System", (Ph.D. Thesis, Stanford Univ.), Tech. Rep., Comp. Sci. Dept., Carnegie-Mellon University.

Newell, A. (1975), "A Tutorial on Speech Understanding Systems", in Reddy (ed.).

Newell, A., Barnett, J., Forgie, J., Green, C., Klatt, D., Licklider, J.C.R., Munson, J., Reddy, D.R., and Woods, W. (1971), "Speech Understanding Systems: Final Report of a Study Group", Reprinted by North-Holland/American Elsevier, Amsterdam.

Reddy, D.R. (ed.) (1975), Speech Recognition, Invited Papers of the IEEE Symposium, Academic Press, N.Y.

Reddy, D.R. (1976), "Speech Recognition by Machine: A Review", to be published, Proc. IEEE, April.

Reddy, D.R., Erman, L.D., and Neely, R.B. (1973), "A Model and a System for Machine Recognition of Speech", IEEE Trans. AU-21, 229-238.

Reddy, D.R., Erman, L.D., Fennell, R.D., and Neely, R.B. (1973), "The HEARSAY-I Speech Understanding System: An Example of the Recognition Process", Proc. 3rd Inter. Joint Conf. on Artificial Intelligence, Stanford, Ca., 185-193, to appear in _IEEE Trans. Computer_, April, 1976.

VanLehn, K.A. (1973), _SAIL Users Manual_, Report STAN-CS-73-373, Computer Science Department, Stanford Univ., Calif.

Vicens, P. (1969), "Aspects of Speech Recognition by Computer", (Ph.D. Thesis, Stanford Univ.), Rept. CS-127, Comp. Sci. Dept., Stanford Univ.

Qtrain Dictionary

"F"	(-,0) [EH1,EH2] F
'S	S
A	(-,0) ([AX1,AX3],[IH1,IH2],AO2 [IH2,IY])
ABOUT	(-,0) (AX2,OW,EH1) N (B1,0) AA2 OW -
ABSOLUTE	(-,0) AE1 - S [AX1,AX2,AX3] L IY [AX1,AX2,AX3] - (T7,0)
ACOUSTIC	(-,0) (AX2,OW,EH1) - [K1,K2] UW S - [T1,T2,T3,T7] (IH1,AX4) - ([K1,K2],0)
ADC	(-,0) IH1 D1 IY S IY
AFRAID	(-,0) AX4 F [ER1,ER2] EH2 IY -
AIRPLANE	(-,0) EH1 [ER1,ER2] - [P1,P2,G] L AE2 IY N
AIRPLANES	(-,0) [AX1,EH1,IH1] [ER1,ER2,ER3] - [P1,P2,G,D1,D2] ([L,ER4],0) ([EH2,OW] IY N (-,0) S,[AX2,AH3] (-,0) [P1,P2])
ALL	(-,0) AO1 L
ALPHA	(-,0) [AA1,AA2,AO1,AH2] L (-,0) [F,K1,T2] [AH1,AX4,AX2,EH2,OW,AX1,AX2,HH,WH,AO1]
AN	(-,0) AX1 N
ANALYSIS	(-,0) AX4 N (OW,AA1) L (IH1,[AX1,AX2]) S (IH1,[AX1,AX2]) S
ANALYZE	(-,0) AX4 N L AA1 IH1 S
AND	(-,0) [AX1,AH3] N (-,0)
ANESTHETIZED	(-,0) AX4 N EH2 S - [T1,T2,T3,T7] [AX1,AX2] S [AX2,AE1,EH2] S -
ANOTHER	(-,0) AX4 N AH4 (-,0) (F,DH) [ER1,ER4]
ARE	(-,0) [AH2,AH4,OW] [ER3,ER2,ER1]
ASTHMA	(-,0) [AX3,AX4,EH2] S M [AX1,AX2,WH]
AT	(-,0) (AE1,EH1) - (T7,0)
ATAL	(-,0) (AE1,AA1) - [T1,T2,T3,T7] AO1 L
ATTACHED	(-,0) [AX2,OW,EH1] - [T1,T2,T3,T7] [AE1,AE2] SH -
AUTOCORRELATION	(-,0) AO1 V [AO2,OW,AX2] - G OW ER1 L EH2 IH2 SH IH2 N
BACK	- (B1,0) AE1 - (K1,0)
BAD	- [B1,P1,D1] [AE1,AE2] -
BAIRSTOW	- ([T2,T3],0) EH2 [ER2,ER3] S - [T6,T3] (AX4,EH2) OW (N,0)
BAIRSTOW'S	- ([T2,T3],0) EH2 [ER2,ER3] S - [T6,T3] (AX4,EH2) OW (N,0) S
BAKER	- [P1,D1] EH2 IY - [K1,G] ER2
BANDWIDTH	- (B1,0) AE1 N - W IH1 - (F,DH)
BECOMES	- [D1,D2] [IH2,IY] - [K2,T2,K1,P1] [AH1,EH2] M (-,0) S
BEEN	- [B1,P1,D1] [IH1,AX1] N
BETA	- ([P1,D2,DH],0) ([IH3,EH2] [IH1,AE2] IY, [AX1,AX3] [IH2,AO2]) ([←1,N],0) [T2,T5,T7,T3] [EH2,AX1,AX2,AX3,HH]
BISHOP	- ([P1,P2],0) IH1 SH [AX4,UW] -
BLACKWELL	- (B1,0) L AE1 - [WH,W] EH1 L
BLEEDING	- [B1,P1,D1] L IY - ([D1,D2,D3],0) [IH1,IH2] [NX,N]
BOUNDARY	- AE1 N ER2 IH2 (N,0)
BY	- (B1,0) AA1 AE1 [AX1,AX2,AX3]
CALCULATE	- [K2,T2,T6] [AE1,AX4] L - ([K2,T2],0) (Y,0) IY [AX1,AX2,AX3] L EH1 IH1 -
CAPTURES	- [K2,T6,T2] [AE1,AX4] P1 SH ER2 S

CASTLE	- [K2,T2,T6] [AE1,AX4] S ([AX1,AX3],0) L
CASTRATED	- [K2,T7,T2] [AE1,AE2] S - [T1,T2,T3,T7] [ER1,ER2] (IH1,AE2) IY - [T1,T2,T3,T7] [AX1,AX2] -
CATEGORY	- [K2,T2,T6] [AE1,AX4] - [T1,T2,T3,T7] (IH1,AX4) (-,0) [G,P1,P2] AO1 [ER1,ER2] (IY, [AX1,AX2])
CEILING	(-,0) S IY L [IH1,IH2] [NX,N]
CENTER	(-,0) S EH2 N (-,0) [T1,T2,T3,T7] [ER1,ER2]
CENTISECONDS	(-,0) S EH2 N - [T1,T2,T3,T7] [AX1,AX2] S EH2 - [K2,T2] [AX1,AX2] N - S
CEPSTRAL	- [K2,T7,T2] EH1 - S - [T1,T2,T3,T7] [ER1,ER2] L
CEPSTRALLY	- [K2,T7,T2] EH1 - S - [T1,T2,T3,T7] [ER1,ER2] L IY
CEPSTRUM	- [K2,T7,T2] EH1 - S - [T1,T2,T3,T7] [ER1,ER2] AH1 M
CHECK	- SH [EH1,EH2] - ([K2,T2,K1],0)
CHEST	- SH [EH1,EH2] S -
CHICKEN-POX	- SH IH1 - [K2,T7,T2] [AX1,AX2] N - [P1,P2] AA1 - S
CHINA	- SH AH1 AE2 N [AX1,AX2,AX3]
CIGARETTES	(-,0) S IH1 - [G,P1,P2] [ER1,ER2] [EH1,EH2] - S
CIRCUMCISED	(-,0) S [AX1,AX2,EH2,AH1] [ER1,ER2] (-,0) [K2,T7,T2,K1] [AX2,EH1] [M,←2] (-,0) S [AX1,AX2] (-,0) S -
CLOUDY	- [K2,T7,T2,K1,G] L (UH,EH2,0) OW (AX2,0) (-,0) ([D1,P3],0) IY
CLUSTERING	- [K2,T7,T2] L AH1 S - [T1,T2,T3,T7] [ER1,ER2] [IH1,IH2] [NX,N]
COEFFICIENTS	- [K2,T7,T2] (UH,EH2) OW [AX1,AX2,AX3] F (IH1,AX4) SH (IH1,AX4) N - S
COMPARE	- [DH,K2,T7,T2] (AH1,[AX1,AX2,AX3]) M - [P1,P2,T2,T3,K2] [EH2,IH1,OW] [ER1,ER2] (N,0)
COMPUTE	- [T2,K1] [ER2,ER4] M - [P1,D1,D2,T4] IH1 (AH3,AX1) (N,0) ([T2,T7],0)
CONSIDER	- [K2,T7,T2] (AH1,[AX1,AX2,AX3]) N S (IH1,AX4) - [D1,D2,D3] [ER1,ER2]
CONSTRUCTION	- [K2,T7,T2] (AH1,[AX1,AX2,AX3]) N S - [T1,T2,T3,T7] [ER1,ER2] AH1 - SH (AX1,IH2,0) N
CONTINUOUS	- [K2,T7,T2] (AH1,[AX1,AX2,AX3]) N - [T1,T2,T3,T7] IH1 N Y IY [AX1,AX2,AX3] S
COVARIANCE	- [K2,T7,T2] (UH,EH2) OW V (AE1,AE2) [ER1,ER2] IY (AE1,AE2) N S
CRAMPS	- [K2,T7,T2] [ER1,ER2] [AE1,AE2] M - S
CUTOFF	- [K2,T7,T2] (AH1,[AX1,AX2,AX3]) - [T1,T2,T3,T7] (AO1,OW) F
CYCLES	(-,0) S AA1 (IH1,[AX1,AX2,AX3,AX4]) - [K2,T7,T2] L S
DB	- [D1,D2,D3] IY - [D1,D1,P1,P2] IY
DEAD	- [D1,D2,D3] [EH1,EH2] -
DEBUGGING	- [D1,D2,D3] IY - [D1,D1,P1,P2] [AX1,AX2,AX3] - [G,P1,P2] [IH1,IH2] [NX,N]
DECIBELS	- [D1,D2,D3] [EH2,AX4] S (IH1,[AX1,AX2,AX3,AX4]) - [D1,D1,P1,P2] [EH2,AX4] L S
DECIMAL	- [D1,T2,T5,D2] [IH3,EH2] S (-,0) M [EH2,AX2] L
DELTA	- ([P1,D1,T5],0) [EH1,EH2,OW,AH2,AA2] L - [P2,T2,T3,T1,T5,T6] [AX1,AX2,AX4,OW]
DERIVATION	- [D1,D2,D3] (AE1,AE2) [ER1,ER2] (IH1,[AX1,AX2,AX3,AX4]) V [EH2,AX4] (IH1,[AX1,AX2,AX3]) SH (AX1,IH2,0) N
DESIGNING	- [D1,D2,D3] (IH1,[AX1,AX2,AX3]) S (AA1,AH1) [EH2,AE1,AX2] N

	[IH1,IH2] [NX,N]
DESIRE	- (D1,0) AX1 S AA1 IH1 ER2
DETAIL	- D1 IY (-,0) [S,T2,T3,T6] [AX1,AX3] L
DIFFERENT	- [D1,D2,D3,G] (IH1,[AX1,AX2,AX3]) [F,T2,K1] [ER1,ER2] N - ([T1,T2,T3,T5],0)
DIGITAL	- [D1,D2,D3] (IH1,[AX1,AX2,AX3]) - SH (IH1,[AX1,AX2,AX3,AX4]) - [T1,T2,T3,T7] L
DISPLAY	- [D1,D3] AX1 S - [P1,P2] L EH1 IH2 (N,0)
DIVIDE	- [D1,D2,P1] AH3 [V,K1] [AA1,AO1] [EH2,AO2,AH3] (N,0) -
DIZZINESS	- [D1,D2,D3] [IH1,AX1,AX2] S IY N [AX1,AX2] S
DO	- [D1,D2,D3] (UW,IH1,IY) (V,0)
DOMAIN	- ([D1,D2,D3],0) (UH,EH2) OW M AE2 IY N
DOUBLE-U	- D3 IH3 AH4 - AX4 Y IY (N,0)
DRINK	- [D1,D2] ER2 IH1 N - (K1,0)
EACH	(-,0) IY - SH
EDITING	(-,0) [EH2,AX4] - [D1,D2,D3] (IH1,[AX1,AX2,AX3,AX4]) - [T1,T2,T3,T7] [IH1,IH2] [NX,N]
EIGHT	(-,0) [AX1,IH2] IY (N,0) - ([T2,T3],0)
EIGHTEEN	(-,0) [AX1,IH2] IY - [T2,T3,T1,T7] IY N
EIGHTY	(-,0) [AX1,IH2] IY - ([D1,D2],0) IY
ELEVEN	(-,0) IY L [EH2,AX4] V [AX1,AX2,AX3] N
END	(-,0) [EH2,AX4] N -
ENHANCEMENT	(-,0) [AX1,AX2,AX3] N (HH,[K1,K2],0) (AE1,AE2,AH3) N S (-,0) M [AX1,AX2,AX3] N - ([T1,T2,T3],0)
EPSILON	(-,0) [AH1,EH2,EH1,OW] (N,0) ([T1,T3],-) S ([AX2,OW,AX1],0) [L,ER4,ER1] ([OW,AH1,AA1] N,[AX1,AX2,AX3])
ESTIMATION	(-,0) [EH2,AX4] S - [T1,T2,T3,T7] (IH1,[AX1,AX2,AX3,AX4]) M [EH2,AX4] (IH1,[AX1,AX2,AX3]) SH (AX1,IH2,0) N
EVER	(-,0) [EH2,AX4] V [ER1,ER2]
FACT	(-,0) F AE1 - ([T1,T2,T3],0)
FACTOR	(-,0) F (AE1,AE2) - [T1,T2,T3,T4,T7] (AO1,OW) [ER1,ER2]
FANT	(-,0) F AA1 N - ([T1,T2,T3],0)
FAST	(-,0) F (AE1,AE2) S - ([T1,T2,T3],0)
FEATURE	(-,0) [F,K1] IY - [T1,T2,T3,T7] [ER1,ER2,ER4]
FEVERISH	(-,0) F IY V [ER1,ER2] IH1 SH
FFT	(-,0) [EH2,AX4] F [EH2,AX4] F - [T1,T2,T3,T7] IY
FIFTEEN	(-,0) F (IH1,[AX1,AX2,AX3,AX4]) F - [T1,T2,T3,T7] IY N
FIFTY	(-,0) F [IH1,AH3,EH2,AX1] F - [T1,T2,T3,T7,D1,D2] IY
FILE	(-,0) [F,K1] [AA1,AO1] AH1 (AX2,0) L
FILTER	(-,0) [F,K1] AX4 L - [T1,T2,T3] [ER1,ER2]
FILTERED	(-,0) F (IH1,[AX1,AX2,AX3,AX4]) L - [T1,T2,T3,T7] [ER1,ER2] -
FIND	(-,0) F (AA1,AH1) [EH2,AE1,AX2] N -
FINDING	(-,0) F (AA1,AH1) [EH2,AE1,AX2] N - [D1,D2,D3] [IH1,IH2] [NX,N]
FIRST	(-,0) [F,K1] ER1 S -
FIVE	(-,0) [F,K1] [AA1,AO1] [EH2,IH2] (AX2,0) [W,V3,K1,P1,P2]
FLOOR	(-,0) F L (AO1,OW) [ER1,ER2]
FOR	(-,0) [K1,F] AO1 [ER1,ER4]
FORMANT	(-,0) [F,K1,T2] [AO1,AO2] [ER1,ER2] M [AE2,IH1,EH2] N - ([T4,T6],0)
FOUR	(-,0) [K1,F] AO2 [ER1,ER4]

FOURIER	(-,0) [F,K1] AH2 AA1 ER3 IH2 IY (N,0)
FOURTEEN	(-,0) F (AO1,OW) [ER1,ER2] - [T1,T2,T3,T7] IY N
FOURTY	(-,0) F (AO1,OW) [ER1,ER2] - [T1,T2,T3,T7] IY
FRANCE	(-,0) [F,K1] [ER1,ER2] [AE1,AE2,IH1] N (-,0) S
FREQUENCY	(-,0) F [ER1,ER2] IY WH [EH2,AX4] N S IY
FREQUENTLY	(-,0) F [ER1,ER2] IY WH [AX1,AX2] N - ([T1,T2,T3,T7],0) L IY
FUNCTION	(-,0) F AH1 [NX,N] - SH (AX1,IH2,0) N
GAMMA	- [K2,T2,D2,T7] [IH1,AE2,AE1,AH3,EH2] [M,P1,←2] [AX2,EH2,OW]
GET	- [G,K1] [IH2,IH3] (N,0) -
GETS	(-,0) [K2,T2] EH2 - S
GIVE	- [T3,T2] IH1 V
GOES	- [G,P1,P2] (UH,EH2) OW [AX1,AX2,AX3] S
GOES-TO	- [G,P1,P2] (UH,EH2) OW [AX1,AX2,AX3] S - [T1,T5,T7] [AX1,←2]
GONORRHEA	- [G,P1,P2] AA1 N [ER1,ER2] IY [AX1,AX2]
GRAMMAR	- G [ER1,ER2,ER3] AE1 M [ER1,ER2]
GRAMMATICAL	- G [ER1,ER2,ER3] (AH4,AX4) M AE1 - [T1,T2,T3,T7] (IH1,AX4) - [K1,K2] (AH2,[AX1,AX2],0) L
GRAPHICS	- G [ER1,ER3] AE1 F [AX1,AX3] - S
GRASS	- [G,P1,P2] [ER1,ER2] [AE1,AE2] S
HAD	(-,0) (HH,0) [AE1,IH3,EH1,EH2] -
HAMMING	(-,0) (HH,[K2,K1,T2,D2],0) (AE1,AE2) M [IH1,IH2] [NX,N]
HANNING	(-,0) (HH,[K1,K2,T2,D2],0) (AE1,AE2) N [IH1,IH2] [NX,N]
HAVE	(-,0) ([HH,K1],0) [AE1,EH2] [V,←2]
HEAD	(-,0) (HH,[K1,K2]) [EH1,EH2] -
HEADACHES	(-,0) [K1,K2] EH2 D1 IH1 - S
HEADLINES	(-,0) [K1,K2,EH2] [AX1,AX4] - [V,P1,K2] [AH1,OW,EH2] [IH1,IY,AH1] (-,0) S
HERTZ	(-,0) (HH,[K1,K2],0) [ER1,ER2] - S
HIGH	(-,0) (HH,[K1,K2],0) (AA1,AH1) [EH2,AE1,AX2]
HIJACKING	(-,0) (HH,[K1,K2]) AH1 IH1 IY - SH [AE1,AE2] - [K1,K2] [IH1,IH2] [NX,N]
HILBERT	(-,0) (HH,[K1,K2],0) (EH1,[AX1,AX2,AX3,AX4]) L - [D1,D1,P1,P2] [ER1,ER2] (N,0) - ([T1,T2,T3],0)
HOSPITALIZED	(-,0) [HH,K1,K2,G,P2] [AA1,AX2] S - [P1,P2,DH] (EH2,[AX1,AX2]) (-,0) [T1,T2,T3,T7,G,P1,P2] ([AX1,AX2],0) L ((AA1,AH1) [EH2,AE1,AX2],[AX2,AX3]) (N,0) (-,0) S -
HOW	(-,0) (HH,[K1,K2]) AA1 W
HUNDRED	(-,0) [T2,P2,K1,HH] [AH1,OW,AX4] N (-,0) (DH,D2,0) (ER2 (N,0),N) -
HYPOTHESIS	(-,0) (HH,[K1,K2],0) (AA1,AH1) [EH2,AE1,AX2] - [P1,P2] AA1 F (IH1,[AX1,AX2,AX3,AX4]) S (IH1,[AX1,AX2,AX3,AX4]) S
I	(-,0) AA1 (AE1,0) [AX1,AX2,AX3]
ILL	(-,0) (IH1,[AX1,AX2]) L
IMAGE	(-,0) (IY,0) (IH1,[AX1,AX2,AX3]) M ([IH1,IH2],[AX1,AX2,AX3,AX4]) - SH
IMAGINARY	(-,0) (IY,0) (IH1,[AX1,AX2,AX3]) M (AE1,AE2) - SH (IH1,[AX1,AX2,AX3,AX4]) N (AE1,AE2) [ER1,ER2] IY
IMMUNIZED	(-,0) [IH1,IH2,AX1,AX2] [M,←2] Y ([IY,IH2],0) UW N [AX1,AX2,AH3,WH] (-,0) S -
IN	(-,0) ([AX1,AX3],IH1,0) N

INCREMENT	(-,0) IH1 N [G,P1,B2] ER3 M IH3 [N,NX] -
INJURED	(-,0) (IH1,[AX1,AX2]) N - SH [ER1,ER2] -
INSTANCE	(-,0) [IH1,IH2] N S - [T2,S] AX1 N S
INTERACTIVE	(-,0) (IH1,IH2) N [T3,T4] [ER1,ER2] AE1 - [D1,D2] IY (N,V)
INTO	(-,0) ([AA1,EH2], [AX3,AX1]) N - [P1,D1,T1,T2,T3,T6] [AX1,AX2,AX3]
INVERSE	(-,0) [IH2,IH1] N V [ER1,ER2] S
IS	(-,0) (N,0) [IY,AX1,AX3,IH2] [S,T6,T4]
ISRAEL	(-,0) [IH2,IY,UW] S [ER1,ER2,V] [AX1,IH2] [OW,AX2] [L,W]
IT	(-,0) [AE2,IH1] - (T7,0)
ITAKURA	(-,0) (IH1,[AX1,AX2,AX3]) - [T1,T2,T3,T7] AH1 - [K2,T7,T2] [ER1,ER2] AH1
JAMES	- [T2,T4,T7] IH1 M S
KING	- [K2,T7,T2] [IH1,IH2] [NX,N]
KNIGHT	(-,0) N AH1 IH1 IY - ([T2,T3,K1],0)
L-I-P	(-,0) [EH1,EH2] L AE1 AE2 - IY
LABELING	(-,0) (L,0) EH1 IH1 [-,B1] (OW,AX4) L [IH1,IH2] [NX,N]
LABELS	(-,0) (L,0) EH1 IH1 [-,B1] OW L S
LEFT	(-,0) L [EH1,EH2] F -
LESIONS	(-,0) L IY SH [AX1,AX2] N S
LET	(-,0) (L,0) AH1 - (T7,0)
LINEAR	(-,0) L (IH1,[AX1,AX2,AX3,AX4]) N IY [ER1,ER2]
LITERAL	(-,0) IH3 V AH2 L
LOGARITHM	(-,0) L (AO1,OW) - [G,P1,P2] ((AE1,AE2),0) [ER1,ER2] (IH1,[AX1,AX2,AX3,AX4]) F M
LONG	(-,0) L AO1 NX
LOOK	(-,0) L (UH,[AX4,OW]) - ([K1,K2],0)
LOW	(-,0) L (UH,EH2) OW
LPC	(-,0) EH1 L - [T1,T2] IY S IY
MARKEL	(-,0) M AA1 [ER1,ER2] - [K2,T7,T2] L
MARKING	(-,0) M ER1 - K2 AX3 [N,NX]
MATE	(-,0) M AE2 IY - ([T2,T7],0)
MAX	(-,0) M AE1 - S
ME	(-,0) M [IH2,IH1]
MEASLES	(-,0) M IY S L S
METHOD	(-,0) M EH2 T2 P1 -
METHODS	(-,0) M [EH2,AX4] F (AH1,[AX1,AX2,AX3]) - S
MICROSECONDS	(-,0) M (AA1,AH1) [EH2,AE1,AX2] - [K2,T7,T2] [ER1,ER2] (UH,EH2) OW S [EH2,AX4] - [K2,T7,T2] [AX1,AX2,AX3] N - S
MILD	(-,0) M AA1 IH1 L -
MILLION	(-,0) M (IH1,[AX1,AX2,AX3,AX4]) L (IH1,[AX1,AX2,AX3,AX4]) [AX1,AX2,AX3] N
MILLISECONDS	(-,0) M (IH1,[AX1,AX2,AX3,AX4]) L (IH1,[AX1,AX2,AX3,AX4]) S [EH2,AX4] - [K2,T7,T2] [AX1,AX2,AX3] N - S
MIN	(-,0) M IH1 N
MINUS	(-,0) M [AA1,EH2,AE1] [IH2,AH3,IY,AE2] N [AX1,IH2] S
MOD	(-,0) M AH1 -
MOVE	(-,0) M AX4 V
MOVES	(-,0) M UW V S
MOVES-TO	(-,0) M UW V S - [T1,T5,T7] [AX1,←2]

MUCH	(-,0) M AA1 - SH
MUMPS	(-,0) M AH1 [M,←2] - S
MURDER	(-,0) [M,←2] [ER1,ER2] [-,D1,B2,M] [ER1,ER2,ER4]
NAUSEA	(-,0) N AO1 SH [AX1,AX2]
NEGAT	(-,0) N [AX1,AX2] - [G,P1,P2] (AE2,IH1) IY - ([T1,T2,T3],0)
NETWORK	(-,0) N IH1 (AX3,0) - W [AX2,ER1,ER4] - (K2,0)
NEWTON	(-,0) N UW - [T1,T2,T3,T7] [AX1,AX2,AX3] N
NINE	(-,0) N (AA1,AH1) [EH2,AE1,AX2] N
NINETEEN	(-,0) N (AA1,AH1) [EH2,AE1,AX2] N - [T1,T2,T3,T7] IY N
NINETY	(-,0) N (AA1,AH1) [EH2,AE1,AX2] N - [T1,T2,T3,T7] IY
NIXON	(-,0) N IH1 - S [AX1,AX2] N
NUMBER	(-,0) N AH1 M [B1,B2] ER1
NUMBNESS	(-,0) N AH1 M N AX1 S
OCTAL	(-,0) (AO1,AA1) - [T1,T2,T3] L
OCTAVE	(-,0) AA1 - [T1,T2,T3,T7] [EH2,AX4] V
OF	(-,0) (OW,AX2) [V,←2] (F,0)
OFTEN	(-,0) AO1 F (- [D1,D2],0) [AX1,AX2] N
ON	(-,0) (AX2,OW) N
ONE	(-,0) W [AH1,AO1,OW,AA1,AX3] N
OPERATION	(-,0) AH1 - [P1,P2] [ER1,ER2] [AE1,AE2] IY SH [AX1,AX2] N
OR	(-,0) [AX2,AH2,AO2] ER4
ORDER	(-,0) (AO1,OW) [ER1,ER2] - [D1,D2,D3] [ER1,ER2]
OVEREAT	(-,0) OW V [ER1,ER2] IY - ([T1,T2,T3],0)
PAIN	- [K1,G,P1] AE2 IY N
PAINS	- [K1,G,P1] AE2 IY N S
PARAMETER	- [B1,G] (AX2,0) [ER2,ER1] [AE2,IH1] M [AX3,AX2,AX4] (-,W) [D2,T1,T2,T3,T7] ER2
PARAMETERS	- [B1,G] (AX2,0) [ER2,ER1] [AE2,IH1] M [AX3,AX2,AX4] (-,W) [D2,T1,T2,T3,T7] ER2 (-,0) S
PART	- [P1,P2] AA1 [ER1,ER2] - ([T1,T2,T3],0)
PASS	- [P1,P2] (AE1,AE2) S
PAWN	- [G,P2] (AO1,OW) (AX2,0) N
PEAK	- [P1,P2] IY - ([K1,K2],0)
PEAKS	- [P1,P2] IY - S
PER	- [P1,P2] [ER1,ER2]
PHONE	(-,0) F (UH,EH2) OW N
PHONEME	(-,0) F (UH,EH2) OW N IY M
PHONETIC	(-,0) [F,T2] AH3 N IH1 [P1,P2] IH2 - ([T2,K2],0)
PHRASE	(-,0) F [ER1,ER2] [EH2,AX4] (IH1,[AX1,AX2,AX3]) S
PICKING	- [P1,P2] (IH1,[AX1,AX2,AX3,AX4]) - [K2,T7,T2] [IH1,IH2] [NX,N]
PITCH	- [G,P2] [IH1,IH2] - SH
PLOT	- [P1,P2] L AA1 - ([T1,T2,T3],0)
PLUS	- [P1,K1,G,P2] (L,0) [AA1,AA2,AH1] S
POINTS	- [P2,K1,G,B2] (HH,0) [AO2,OW] [ER1,ER2,AX2] N - S
POSITION	- [K1,G] AX1 S UW SH (AX1,0) N
POSITIONS	- [K1,G] AX1 S UW SH (AX1,0) N S
POST-EMPHASIS	- [P1,P2] (UH,EH2) OW S - ([T1,T2,T3,T7],0) [EH2,AX4] M F AH1 S (IH1,[AX1,AX2,AX3,AX4]) S
POT	- [P1,P2] AA1 - ([T1,T2,T3],0)
POWER	- [P1,K1,G] AO1 W [ER1,ER2]

PRE-EMPHASIS	- [P1,P2] [ER1,ER2] IY [EH2,AX4] M F AH1 S (IH1,[AX1,AX2,AX3,AX4]) S
PREDICTION	- [P1,P2] [ER1,ER2] IY - [D1,D2,D3] (IH1,[AX1,AX2,AX3,AX4]) - SH (AX1,IH2,0) N
PREDICTIVE	- [P1,P2] [ER1,ER2] [AX1,AX2,AX3] - [D1,D2,D3] (IH1,[AX1,AX2,AX3,AX4]) - [T1,T2,T3,T7] (IH1,[AX1,AX2,AX3,AX4]) V
PRESENT	- P1 [ER1,ER3] EH1 S AH3 N -
PRONY	- [P1,P2] [ER1,ER2] ((UH,EH2),0) OW N IY
PROTOCOL	- ([G,K1],0) [ER1,ER3] UH OW V AX1 - G AO1 L
PUT	- [D1,G,P3,P2] [EH1,EH2,AX3,OW] (N,0) -
QUEEN	(-,0) WH (IH1,0) IY N
RABINER	(-,0) [ER1,ER2] AH1 - [D1,D1,P1,P2] (IH1,[AX1,AX2,AX3,AX4]) N [ER1,ER2]
RAPE	(-,0) ER4 IH1 IY (N,0) - (K1,0)
RATING	(-,0) [ER1,ER2] [EH2,AX4] (IH1,[AX1,AX2,AX3]) - [T1,T2,T3,T7] [IH1,IH2] [NX,N]
REAL	(-,0) [ER1,ER2] IY L
RECTANGULAR	(-,0) [ER1,ER2] [EH2,AX4] - [T1,T2,T3,T7] [EH2,AX4] (IH1,[AX1,AX2,AX3]) [NX,N] - [G,P1,P2] Y UW L (AA1,0) [ER1,ER2]
REQUEST	(-,0) [ER2,ER4] [IH2,AX1] WH (W,0) EH1 S -
RESOLUTION	(-,0) [ER1,ER2] [EH2,AX4] S (UH,EH2) OW L UW SH (AX1,IH2,0) N
RIGHT	(-,0) ER1 UH IH1 - (T4,0)
ROBINSON	(-,0) [ER1,ER2] AA1 - [D1,D1,P1,P2] (IH1,[AX1,AX2,AX3,AX4]) N S AH1 N
ROOK	(-,0) [ER1,ER4] [UH,AA2] - (K1,0)
ROOT	(-,0) [ER1,ER2] UW - ([T1,T2,T3],0)
ROOTS	(-,0) [ER3,ER4] UW - S
RUSSIA	(-,0) [ER1,ER2] [AX1,AX2] SH [AX1,AX2]
SAY	(-,0) S IH1 IY
SCALE	(-,0) S - [K2,T7,T2] [EH2,AX4] (IH1,[AX1,AX2,AX3]) L
SCHAFFER	(-,0) SH [EH2,AX4] (IH1,[AX1,AX2,AX3]) F [ER1,ER2]
SECOND	(-,0) S [EH2,AX4] - [K2,T7,T2] AH1 N -
SEGMENT	(-,0) S AX4 - M AX1 N -
SENTENCE	(-,0) S EH2 N [P1,T2,T6] AX1 (N,0) S
SERIOUS	(-,0) S IH1 [ER1,ER2] IY [AX1,AX2] S
SEVEN	(-,0) S [EH1,EH2] [-,V,T3,K1,P2] ([AX1,AX3,EH1,EH2] N, [AX1,AX2,AX3])
SEVENTEEN	(-,0) S [EH2,AX4] V [EH2,AX4] N - [T1,T2,T3,T7] IY N
SEVENTY	(-,0) S [EH2,AX4] V [EH2,AX4] N - [T1,T2,T3,T7] IY
SEVERE	(-,0) S [AX1,HH] (-,0) [V,←2] IY ([AX1,AX3],0) [ER2,ER4,P2]
SEX	(-,0) S [AX4,EH1,AX3] - S
SHARP	(-,0) SH AH1 [ER1,ER2] -
SHOW	(-,0) SH (UH,EH2) OW
SICK	(-,0) S IH1 - ([K1,K2],0)
SIDE	(-,0) S AH1 AE2 -
SIMULATION	(-,0) S (IH1,[AX1,AX2,AX3,AX4]) M Y UW L [EH2,AX4] (IH1,[AX1,AX2,AX3]) SH (AX1,IH2,0) N
SIX	(-,0) S (IH1,AX4) - S

SIXTEEN	(-,0) S (IH1,[AX1,AX2,AX3,AX4]) - S - [T1,T2,T3,T7] IY N
SIXTY	(-,0) S (IH1,[AX1,AX2,AX3,AX4]) - S - [T1,T2,T3,T7] IY
SMOKE	(-,0) S M UH OW AX2 - ([K1,T2],0)
SMOOTHED	(-,0) S (-,0) M UW V - DH -
SMOOTHING	(-,0) S M UW (F,DH) [IH1,IH2] [NX,N]
SPEAKER	(-,0) S - [D1,D2,P1] IY - [K1,K2] [ER2,ER1]
SPECIFICATION	(-,0) S - [AX4,EH2] S [AX1,AX2,AX3] F [AX1,AX2,AX3] - K1 [AX1,IH2] SH [AX1,AX2,AX3]
SPECTRAL	(-,0) S - [P1,P2] [EH2,AX4] - [T1,T2,T3,T4,T7] [ER1,ER2] L
SPECTROGRAM	(-,0) S - [P1,P2] [EH1,IH3] - [T1,T2,T3,T4,T7] [ER1,ER2] ([UH,OW],0) - [G,P1,P2] [ER1,ER2] [EH2,IH1] [M,←2]
SPECTRUM	(-,0) S - P1 EH1 - [T2,T4,T7] [ER1,ER4] M
SPEECH	(-,0) S - [P1,P2] IY - SH
START	(-,0) S - [T1,T2,T3,T7] AA1 [ER1,ER2] - ([T1,T2,T3],0)
STARTING	(-,0) S - [T1,T2,T3,T7] AA1 [ER1,ER2] - [T1,T2,T3,T7] [IH1,IH2] [NX,N]
STEPS	(-,0) S - [S,T1,T6] IH3 (N,0) - S
STOP	(-,0) S - (T6,0) AA1 -
STORE	(-,0) S - [T1,T4,T5,T6] (AO1,OW) [ER1,ER4]
STORIES	(-,0) S - [T1,T4,T5,T6] (AO1,OW) [ER1,ER4] IY (-,0) S
SUMMARY	(-,0) S AH1 M ER4
SURGERY	(-,0) S [ER1,ER2] - SH [ER1,ER2] IY
SYMBOL	(-,0) S [IH1,EH2] M B1 (UH,EH2) L
SYNTHESIS	(-,0) S (IH1,[AX1,AX2,AX3,AX4]) N F [AX1,AX2,AX3] S (IH1,[AX1,AX2,AX3,AX4]) S
TAKE	- [T1,T2,T3,T7] [EH2,AX4] (IH1,[AX1,AX2,AX3]) - ([K1,K2],0)
TAKES	- [T1,T2,T5,T7] AE2 IY - S
TASK	- (S,T2) AE1 S - ([K1,T2],0)
TELL	- [T2,T5,T6,T1] [EH1,AO1] L
TEN	- [T1,T2,T3,T7] [EH2,AX4] N
TESTING	- [T1,T2,T3,T7] [EH2,AX4] S - [T1,T2,T3,T7] [IH1,IH2] [NX,N]
THE	(-,0) (DH,[T1,T3,F,T2,G,K1]) ([AX1,AX2,AH3],IY)
THIRTEEN	(-,0) F [ER1,ER2] - [T1,T2,T3,T7] IY N
THIRTY	(-,0) [G,T5,F] (AX4,0) [ER1,ER2] (- [T1,T2,T3,T7],M) IY (N,0)
THOUSAND	(-,0) F (UH,EH2) OW S (AE1,AE2) N -
THREE	(-,0) [F,T2,T6] [ER1,ER2,ER4] (IY,[AX1,AX3]) (N,0)
TIME	- [T1,T2,T3,T7] AA1 IH1 M
TIMES	- [T1,T2,T3,T7,D1,K1,P2] [AA1,EH2,OW] [AH3,AX1,AX3,IH1] [←2,M] S
TO	- [T1,T5,T7,T6] [AX1,←2] (V,0)
TRACKING	- [T2,T4,K2] [ER1,ER2,ER4] EH2 - [T2,K1,K2] (IH2,AX3) (NX,N)
TRACKS	- [T1,T2,T3,T7] [ER1,ER2] (AE1,AE2) - S
TRANSCRIPTION	- [T2,T4] [ER1,ER2,ER4] EH2 N S - [K1,K2] [ER1,ER2,ER4] [IH1,IH2] - SH (AX1,0) N
TRANSFORM	- [T1,T2,T3,T7,K2] [ER1,ER2] (AE1,AE2,IH1) N S (-,0) [F,K1] ((AO1,OW) [ER1,ER2] M,[L,AX2] M)
TRIANGULAR	- [T1,T2,T3,T7] [ER1,ER2] (AA1,AH1) [EH2,AE1,AX2] [EH2,AX4] (IH1,[AX1,AX2,AX3]) N - [G,P1,P2] Y UW L (AA1,0) [ER1,ER2]
TUBERCULOSIS	- [T1,T2,T3,T7] UW - [B1,B2] [ER1,ER2] -

	[K2,T7,T2] Y UW L OW S [AX1,AX2] S
TWELVE	- [T2,K1,K2] W [EH1,OW] L [W,V]
TWENTY	- [T1,T2,T3,T7] W [EH2,AX4] N - [T1,T2,T3,T7] IY
TWO	- [T1,T4,T5] (IH1 UW,AX3) (V,0)
UNTIL	(-,0) [HH,ER4,AX2] N (-,0) [S,T6,T5] EH2 L
URINE	(-,0) Y [ER1,ER2] [AX1,AX2] N
US	(-,0) (AO1,0) EH2 S
USE	(-,0) Y UW S
USING	(-,0) Y UW S IH2 [N,NX]
UTTERANCE	(-,0) [AA2,AA1] (-,0) ([T2,T7],HH) ([AX2,AX3],0) [ER1,ER3] ([AX1,AX3],0) N S
VALUE	(-,0) V (AE1,AE2) L Y UW
VIETNAM	(-,0) [V,T2] IY [IH1,IH2,AX1,AX3] - N [EH2,ER2,AX2] [M,P1,P3]
WANT	(-,0) W [AO1,AA2,AH1] N - (T7,0)
WAR	(-,0) W AO2 [ER1,ER2]
WATERGATE	(-,0) W AO2 - ([T1,T2,D1,D2],0) [ER1,ER2] - [G,P1,P2] AE2 IY - ([T1,T2],0)
WAVEFORM	(-,0) W IH1 V3 F AX2
WEIGH	(-,0) W (IH1,AE2) IY
WERE	(-,0) W [ER1,ER2]
WHAT	(-,0) W [AH1,AH4] - ([T1,T2],0)
WHEN	(-,0) W [AX1,AX2] N
WHERE	(-,0) W IH1 [ER1,ER2]
WINDOW	(-,0) W IH1 N D1 EH1 (AH2,AX2,OW) (W,0)
WITH	(-,0) (W,HH) (IH1,[AX1,AX2,AX3,AX4]) (F,W)
WORD	(-,0) W ER1 -
YOU	(-,0) (Y,T2) IY [AX1,AX2,AX3]
YOUR	(-,0) [IY,OW] [ER1,ER2] (ER4,0)
ZERO	(-,0) S [IY,IH2] [ER1,ER2] (OW,0) AX2
[	-
]	-

Desk Calc Dictionary

[	-
]	-
ABSOLUTE	(-,0) (HH,0) AE (← (-,0),-) S (IX,0) L UW ((←,-) (- T,0),DX)
ALPHA	(-,0) (HH,0) (AE,AA) EL (← (-,0),0) F (AX,AH2)
BECOMES	(← (-,0),-) (B,0) (IH,IX,IY4) (← (-,0),-) K AH3 M (Z (S,0),S)
BETA	(← (-,0),-) (B,0) E1 E2 E3 (DX,(← (-,0),-)(T,0)) (AX,AH2,IH5)
DECIMAL	(← (-,0),-) (D,DT) EH S (-,0) M EL2
DELTA	(← (-,0),-) (D,DT) EH EL (← (-,0),-) (T,DX) (AX,AH2,IH5)
DIVIDE	(← (-,0),-) (D,DT) (AX,IX) (VX,V) (F,0) Y1 Y2 Y3 (← (-,0),-) (D IX,0)
EIGHT	(-,0) (HH,0) E1 E2 E3 ((←,-) (- T,0),DX)
EPSILON	(-,0) (HH,0) EH (← (-,0),-) S (-,0) (IX,IH3,0) L AO N
FACT	(-,0) F AE ((←,-) (- T,0),DX)
FIVE	(-,0) F Y1 Y2 Y3 V
FOUR	(-,0) F AH4 ER
GAMMA	(← (-,0),-) G AE M (AX,AH2)
GETS	(← (-,0),-) G IH (← (-,0),-) S
IN	(-,0) (HH,0) (IX!,IH2!) N
INTO	(-,0) (HH,0) (IX!,IH2!) N (←,0) (-,0) T (AX,IX,UW2)
IS	(-,0) (HH,IX) (IH,IH4) (Z (S,0),S)
MAX	(-,0) M AE (← (-,0),-) S
MIN	(-,0) M (IH,IX) N
MINUS	(-,0) M Y1 Y2 Y3 N IX (HH,0) S
MOD	(-,0) M AA3 (← (-,0),-)
NEGATE	(-,0) N (AX,E1 E2 E3,EH3) (← (-,0),-) G E1 E2 E3 ((←,-) (- T,0),DX)
NINE	(-,0) N Y1 Y2 Y3 N
OCTAL	(-,0) (HH,0) AA (← (-,0),-) T EL2
ONE	(-,0) W AH N
PLUS	(← (-,0),-) P (EL3,EL,L) AH5 S
POWER	(← (-,0),-) P AW1 AW2 AW3 ER
PUT	(← (-,0),-) P UH ((←,-) (- T,0),DX)
SEVEN	(-,0) S (HH,0) EH (VX,V) (-,0) (IX,EH,AX) N
SHOW	(-,0) SH AH6 OW OW2
SIX	(-,0) S IH2 (← (-,0),-) S
STORE	(-,0) S - T AH ER
THREE	(← (-,0),-) TH R IY IY2
TIMES	(← (-,0),-) T Y1 Y2 Y3 M (-,0) (Z (S,0),S)
TWO	(← (-,0),-) T (IH,0) UW
WHAT	(-,0) W AA2 ((←,-) (- T,0),DX)
ZERO	(-,0) Z (IY4,IH) (R,ER) OW (OW2,0)

Desk Calc BNF Grammar

```
<REQUEST>::=          [ <COMMAND> ]

<COMMAND>::=          <SET-WORD> <SIMPLE-EXPRE> <IN-WORD> <VARIABLEDF>
                      <VARIABLE> <GET-WORD> <SIMPLE-EXPRF>
                      <SHOW-WORD> <SIMPLE-EXPRF>

<SET-WORD>::=         STORE
                      PUT

<IN-WORD>::=          IN
                      INTO

<GET-WORD>::=         GETS
                      BECOMES

<SHOW-WORD>::=        WHAT IS
                      SHOW

<BIN-OPE>::=          PLUS
                      MINUS
                      TIMES
                      DIVIDE
                      MOD
                      POWER
                      MAX
                      MIN

<UN-OPE>::=           NEGATE
                      ABSOLUTE
                      FACT

<BIN-OPF>::=          PLUS
                      MINUS
                      TIMES
                      DIVIDE
                      MOD
                      POWER
                      MAX
                      MIN

<UN-OPF>::=           NEGATE
                      ABSOLUTE
                      FACT

<SIMPLE-EXPRE>::=     <PRIMARYCE> <BIN-OPE> <PRIMARYDE>
                       <UN-OPE> <PRIMARYDE>
```

```
                          <PRIMARYDE>

<VARIABLECE>::=           ALPHA
                          BETA
                          GAMMA
                          DELTA
                          EPSILON

<PRIMARYCE>::=            <RADIXCE> <INTEGERCE>
                          <INTEGERCE>
                          <VARIABLECE>

<RADIXCE>::=              OCTAL
                          DECIMAL

<INTEGERCE>::=            <DIGITACE> <INTEGERCE2>
                          <DIGITACE>

<DIGITACE>::=             ZERO
                          ONE
                          TWO
                          THREE
                          FOUR
                          FIVE
                          SIX
                          SEVEN
                          EIGHT
                          NINE

<INTEGERCE2>::=           <DIGITACE2><INTEGERCE>
                          <DIGITACE2>

<DIGITACE2>::=            ZERO
                          ONE
                          TWO
                          THREE
                          FOUR
                          FIVE
                          SIX
                          SEVEN
                          EIGHT
                          NINE

<VARIABLEDE>::=           ALPHA
                          BETA
                          GAMMA
                          DELTA
                          EPSILON

<VARIABLE>::=             ALPHA
```

```
                        BETA
                        GAMMA
                        DELTA
                        EPSILON

<PRIMARYDE>::=          <RADIXDE> <INTEGERDE>
                        <INTEGERDE>
                        <VARIABLEDE>

<RADIXDE>::=            OCTAL
                        DECIMAL

<INTEGERDE>::=          <DIGITADE> <INTEGERDE2>
                        <DIGITADE>

<DIGITADE>::=           ZERO
                        ONE
                        TWO
                        THREE
                        FOUR
                        FIVE
                        SIX
                        SEVEN
                        EIGHT
                        NINE

<INTEGERDE2>::=         <DIGITADE2><INTEGERDE>
                        <DIGITADE2>

<DIGITADE2>::=          ZERO
                        ONE
                        TWO
                        THREE
                        FOUR
                        FIVE
                        SIX
                        SEVEN
                        EIGHT
                        NINE

<SIMPLE-EXPRF>::=       <PRIMARYCF> <BIN-OPF> <PRIMARYDF>
                         <UN-OPF> <PRIMARYDF>
                         <PRIMARYDF>

<VARIABLECF>::=         ALPHA
                        BETA
                        GAMMA
                        DELTA
                        EPSILON
```

```
<PRIMARYCF>::=        <RADIXCF> <INTEGERCF>
                      <INTEGERCF>
                      <VARIABLECF>

<RADIXCF>::=          OCTAL
                      DECIMAL

<INTEGERCF>::=        <DIGITACF> <INTEGERCF2>
                      <DIGITACF>

<DIGITACF>::=         ZERO
                      ONE
                      TWO
                      THREE
                      FOUR
                      FIVE
                      SIX
                      SEVEN
                      EIGHT
                      NINE

<INTEGERCF2>::=       <DIGITACF2><INTEGERCF>
                      <DIGITACF2>

<DIGITACF2>::=        ZERO
                      ONE
                      TWO
                      THREE
                      FOUR
                      FIVE
                      SIX
                      SEVEN
                      EIGHT
                      NINE

<VARIABLEDF>::=       ALPHA
                      BETA
                      GAMMA
                      DELTA
                      EPSILON

<PRIMARYDF>::=        <RADIXDF> <INTEGERDF>
                      <INTEGERDF>
                      <VARIABLEDF>

<RADIXDF>::=          OCTAL
                      DECIMAL

<INTEGERDF>::=        <DIGITADF> <INTEGERDF2>
                      <DIGITADF>
```

```
<DIGITADF>::=          ZERO
                       ONE
                       TWO
                       THREE
                       FOUR
                       FIVE
                       SIX
                       SEVEN
                       EIGHT
                       NINE

<INTEGERDF2>::=        <DIGITADF2><INTEGERDF>
                       <DIGITADF2>

<DIGITADF2>::=         ZERO
                       ONE
                       TWO
                       THREE
                       FOUR
                       FIVE
                       SIX
                       SEVEN
                       EIGHT
                       NINE
```

QTRAIN Juncture Rules

```
0,<←
[EH2,AH3,AX3,AX1,IY],[ER1,ER4];(←2,0)
0,[L,W];(←2,0)
```

Desk Calc Juncture Rules

```
ER,#↑(.<,.R)(.AXX,0)
N,#↑(.<,0) (.AXX,0) (.←,0)
V,#↑((.<,.VX) (.F,0),(.<,0) ←)
M,N{
N,M{
0,<{
[AA,AA2,AA3,AE,AH,AH2,AH3,AH4,AH5,AH6,AO,AW3],#↑(.AXX,.<)(.AXX,0)(.←,0)
[OW,OW2,UH,UW,AX,EH,EH3,EL,EL2,ER],#↑(.AXX,.<)(.AXX,0)(.←,0)
[IH,IH2,IH3,IH4,IH5,IX,IY,IY2,IY4,E3,Y3],#↑(.IXX,.<)(.IXX,0)(.←,0)
DX,#←
DX,[-,B,D,DX,F,G,HH,K,P,S,SH,T,TH,V,Z,←]←
S,F↑(.< (-,0) (>,F2))
```

Desk Calc Phonemes and Durations

PHONEME	MIN	MAX
-	3	1000
AA	5	20
AA2	5	15
AA3	5	15
AE	5	15
AH	5	15
AH2	5	16
AH3	5	15
AH4	3	8
AH5	5	30
AH6	5	15
AO	5	15
AW1	5	15
AW2	5	15
AW3	5	15
AX	3	10
AXX	1	10
B	1	2
D	1	3
DT	1	3
DX	1	4
E1	2	8
E2	2	8
E3	4	8
EH	3	15
EH3	3	15
EL	4	12
EL2	4	12
EL3	1	2
ER	4	20
F	5	20
F2	1	20
G	1	5
HH	2	9
IH	5	25
IH2	5	25
IH3	2	6
IH4	20	40
IH5	4	12
IX	3	10
IXX	1	10
IY	5	15
IY2	5	15
IY4	5	15
K	3	8

L	2	10
M	2	10
N	3	20
OW	3	15
OW2	5	15
P	2	8
R	3	10
S	6	35
SH	6	15
T	1	12
TH	2	18
UH	5	15
UW	5	15
UW2	4	10
V	2	10
VX	2	10
W	1	12
Y1	3	8
Y2	3	15
Y3	2	15
Z	3	15
←	2	8

Test Summary of Speaker BL, No Syntax, on Test Data recorded with Training Data.

UTT	LNG	WORDS	#COR	%COR	R-TIME	#STATES	SEGS
1	2.0	5	4	80.0	8.9	44	62
2	1.6	5	4	80.0	9.4	53	49
3	1.5	4	4	100.0	9.3	54	44
4	2.0	5	5	100.0	8.7	42	63
5	1.7	5	5	100.0	10.0	50	58
6	1.9	5	5	100.0	8.6	45	60
7	2.2	6	6	100.0	10.2	61	74
8	2.0	6	5	83.3	9.1	56	62
9	2.5	7	7	100.0	8.2	44	72
10	2.7	8	6	75.0	7.9	39	76
11	2.3	7	6	85.7	9.6	54	63
12	2.3	6	6	100.0	9.2	32	70
13	2.6	7	5	71.4	11.0	58	86
14	2.0	4	1	25.0	11.7	67	61
15	1.8	4	4	100.0	12.0	64	62
16	2.0	4	3	75.0	8.1	50	54
17	1.7	5	5	100.0	8.5	48	45
18	1.5	4	4	100.0	6.3	29	36
19	2.4	7	7	100.0	10.2	48	80
20	2.2	6	5	83.3	11.1	55	67

LNG	= Utterance length in seconds
WORDS	= Number of words in utterance
#COR	= Number of words correctly recognized
%COR	= % of words correctly recognized
R-TIME	= Number of times real time needed for recognition Add 8.4 for ACS and LPC calculations.
#STATES	= Average number of states checked at each segment
SEGS	= Number of segments produced (LNGx100 = number of 10ms. time samples)

Total words =	110
Correct words =	97 = 88.2%
Correct utterances =	11 = 55.0%
Average search time =	9.4 times real time
Including ACS and LPC calculation =	17.8 times real time
Average number of states searched =	49
Ratio of segments/10ms samples =	30.6%

Test Detail of Speaker BL, No Syntax, on Test Data recorded with Training Data.

```
1   UTT="STORE NEGATE ALPHA INTO EPSILON"
    REC="STORE EIGHT BETA ALPHA INTO EPSILON"                 CORRECT=4/5
2   UTT="PUT ABSOLUTE BETA IN DELTA"
    REC="PUT ABSOLUTE EIGHT IN DELTA"                         CORRECT=4/5
3   UTT="ALPHA GETS FACT GAMMA"
    REC="ALPHA GETS FACT GAMMA"                               CORRECT=4/4
4   UTT="BETA BECOMES OCTAL ONE ZERO"
    REC="BETA BECOMES OCTAL ONE ZERO"                         CORRECT=5/5
5   UTT="GAMMA GETS DECIMAL TWO THREE"
    REC="GAMMA GETS DECIMAL TWO THREE"                        CORRECT=5/5
6   UTT="DELTA BECOMES FOUR PLUS FIVE"
    REC="DELTA BECOMES FOUR PLUS FIVE"                        CORRECT=5/5
7   UTT="EPSILON GETS SIX EIGHT MINUS DELTA"
    REC="EPSILON GETS SIX EIGHT MINUS DELTA"                  CORRECT=6/6
8   UTT="WHAT IS SEVEN NINE TIMES EPSILON"
    REC="WHAT GETS SEVEN NINE TIMES EPSILON"                  CORRECT=5/6
9   UTT="SHOW ONE ZERO TWO DIVIDE THREE FOUR"
    REC="SHOW ONE ZERO TWO DIVIDE THREE FOUR"                 CORRECT=7/7
10  UTT="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"
    REC="STORE FIVE SIX NINE SEVEN INTO ALPHA"                CORRECT=6/8
11  UTT="PUT NINE POWER TWO ONE IN BETA"
    REC="PUT IN NINE POWER TWO ONE IN BETA"                   CORRECT=6/7
12  UTT="ALPHA BECOMES THREE ZERO MAX GAMMA"
    REC="ALPHA BECOMES THREE ZERO MAX GAMMA"                  CORRECT=6/6
13  UTT="BETA GETS DECIMAL FOUR SIX MIN DELTA"
    REC="MIN GETS DECIMAL FOUR SIX IN IN DELTA"               CORRECT=5/7
14  UTT="GAMMA BECOMES NEGATE EPSILON"
    REC="IN MOD BECOMES MIN BETA EPSILON"                     CORRECT=1/4
15  UTT="DELTA GETS ABSOLUTE ALPHA"
    REC="DELTA GETS ABSOLUTE ALPHA"                           CORRECT=4/4
16  UTT="EPSILON BECOMES FACT BETA"
    REC="EPSILON BECOMES FACT BETA IN"                        CORRECT=3/4
17  UTT="WHAT IS SEVEN PLUS EIGHT"
    REC="WHAT IS SEVEN PLUS EIGHT"                            CORRECT=5/5
18  UTT="SHOW NINE MINUS FIVE"
    REC="SHOW NINE MINUS FIVE"                                CORRECT=4/4
19  UTT="STORE TWO ZERO TIMES THREE INTO GAMMA"
    REC="STORE TWO ZERO TIMES THREE INTO GAMMA"               CORRECT=7/7
20  UTT="PUT FOUR DIVIDE DELTA IN EPSILON"
    REC="PUT FOUR DIVIDE DELTA IN IN EPSILON"                 CORRECT=5/6
```

Test Summary of Speaker KP, No Syntax, on Test Data recorded with Training Data.

UTT	LNG	WORDS	#COR	%COR	R-TIME	#STATES	SEGS
1	2.1	5	5	100.0	8.7	57	55
2	1.9	5	5	100.0	10.7	80	52
3	1.7	4	4	100.0	10.3	71	46
4	2.1	5	5	100.0	9.4	52	62
5	1.9	5	5	100.0	10.6	70	53
6	2.0	5	3	60.0	9.6	61	53
7	2.4	6	6	100.0	11.3	82	69
8	2.4	6	5	83.3	8.5	61	58
9	2.7	7	5	71.4	9.2	57	73
10	3.2	8	8	100.0	9.0	60	85
11	2.4	7	7	100.0	9.1	68	61
12	2.5	6	6	100.0	7.1	36	64
13	2.6	7	5	71.4	11.7	78	82
14	2.1	4	4	100.0	8.4	55	56
15	1.8	4	4	100.0	9.5	66	52
16	2.0	4	4	100.0	7.7	57	44
17	1.7	5	5	100.0	9.7	70	41
18	1.7	4	2	50.0	5.2	28	31
19	2.8	7	7	100.0	8.6	55	74
20	2.4	6	6	100.0	10.8	73	68

LNG	= Utterance length in seconds
WORDS	= Number of words in utterance
#COR	= Number of words correctly recognized
%COR	= % of words correctly recognized
R-TIME	= Number of times real time needed for recognition ADD 8.4 for ACS and LPC calculations.
#STATES	= Average number of states checked at each segment
SEGS	= Number of segments produced (LNGx100 = number of 10ms. time samples)

Total words =	110
Correct words =	101 = 91.8%
Correct utterances =	15 = 75.0%
Average search time =	9.3 times real time
Including ACS and LPC calculation =	17.7
Average number of states searched =	61
Ratio of segments/10ms samples =	26.7%

Test Detail of Speaker KP, No Syntax, on Test Data recorded with Training Data.

```
1   UTT="STORE NEGATE ALPHA INTO EPSILON"
    REC="STORE NEGATE ALPHA INTO EPSILON"                       CORRECT=5/5
2   UTT="PUT ABSOLUTE BETA IN DELTA"
    REC="PUT ABSOLUTE BETA IN DELTA"                            CORRECT=5/5
3   UTT="ALPHA GETS FACT GAMMA"
    REC="ALPHA GETS FACT GAMMA"                                 CORRECT=4/4
4   UTT="BETA BECOMES OCTAL ONE ZERO"
    REC="BETA BECOMES OCTAL ONE ZERO"                           CORRECT=5/5
5   UTT="GAMMA GETS DECIMAL TWO THREE"
    REC="GAMMA GETS DECIMAL TWO THREE"                          CORRECT=5/5
6   UTT="DELTA BECOMES FOUR PLUS FIVE"
    REC="DELTA BECOMES FOUR EPSILON"                            CORRECT=3/5
7   UTT="EPSILON GETS SIX EIGHT MINUS DELTA"
    REC="EPSILON GETS SIX EIGHT MINUS DELTA"                    CORRECT=6/6
8   UTT="WHAT IS SEVEN NINE TIMES EPSILON"
    REC="WHAT GETS SEVEN NINE TIMES EPSILON"                    CORRECT=5/6
9   UTT="SHOW ONE ZERO TWO DIVIDE THREE FOUR"
    REC="SHOW IN ZERO TWO TWO FIVE THREE FOUR"                  CORRECT=5/7
10  UTT="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"
    REC="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"             CORRECT=8/8
11  UTT="PUT NINE POWER TWO ONE IN BETA"
    REC="PUT NINE POWER TWO ONE IN BETA"                        CORRECT=7/7
12  UTT="ALPHA BECOMES THREE ZERO MAX GAMMA"
    REC="ALPHA BECOMES THREE ZERO MAX GAMMA"                    CORRECT=6/6
13  UTT="BETA GETS DECIMAL FOUR SIX MIN DELTA"
    REC="BETA GETS DECIMAL FOUR EIGHT SIX MIN DELTA IN"         CORRECT=5/7
14  UTT="GAMMA BECOMES NEGATE EPSILON"
    REC="GAMMA BECOMES NEGATE EPSILON"                          CORRECT=4/4
15  UTT="DELTA GETS ABSOLUTE ALPHA"
    REC="DELTA GETS ABSOLUTE ALPHA"                             CORRECT=4/4
16  UTT="EPSILON BECOMES FACT BETA"
    REC="EPSILON BECOMES FACT BETA"                             CORRECT=4/4
17  UTT="WHAT IS SEVEN PLUS EIGHT"
    REC="WHAT IS SEVEN PLUS EIGHT"                              CORRECT=5/5
18  UTT="SHOW NINE MINUS FIVE"
    REC="SHOW NINE EPSILON"                                     CORRECT=2/4
19  UTT="STORE TWO ZERO TIMES THREE INTO GAMMA"
    REC="STORE TWO ZERO TIMES THREE INTO GAMMA"                 CORRECT=7/7
20  UTT="PUT FOUR DIVIDE DELTA IN EPSILON"
    REC="PUT FOUR DIVIDE DELTA IN EPSILON"                      CORRECT=6/6
```

Test Summary of Speaker DS, No Syntax, on Test Data recorded with Training Data.

UTT	LNG	WORDS	#COR	%COR	R-TIME	#STATES	SEGS
1	2.0	5	4	80.0	10.5	60	61
2	2.0	5	5	100.0	10.3	76	57
3	1.8	4	4	100.0	10.7	80	55
4	2.4	5	4	80.0	8.1	42	68
5	2.2	5	5	100.0	9.6	61	60
6	2.2	5	4	80.0	8.6	45	60
7	2.6	6	6	100.0	9.6	58	71
8	2.4	6	5	83.3	8.5	60	65
9	3.3	7	7	100.0	8.9	64	89
10	3.5	8	7	87.5	9.4	55	93
11	2.6	7	7	100.0	10.0	62	70
12	2.9	6	6	100.0	9.2	53	87
13	3.1	7	6	85.7	10.1	63	91
14	2.1	4	4	100.0	9.5	58	62
15	1.9	4	4	100.0	10.9	78	57
16	2.4	4	4	100.0	8.7	61	62
17	1.9	5	4	80.0	8.9	69	48
18	1.9	4	4	100.0	5.9	32	42
19	3.0	7	6	85.7	10.1	60	94
20	2.5	6	5	83.3	10.7	68	76

LNG	= Utterance length in seconds
WORDS	= Number of words in utterance
#COR	= Number of words correctly recognized
%COR	= % of words correctly recognized
R-TIME	= Number of times real time needed for recognition ADD 8.4 for ACS and LPC calculations.
#STATES	= Average number of states checked at each segment
SEGS	= Number of segments produced (LNGx100 = number of 10ms. time samples)

Total words =	110
Correct words =	101 = 91.8%
Correct utterances =	11 = 55.0%
Average search time =	9.4 times real time
Including ACS and LPC calculation =	17.8
Average number of states searched =	60
Ratio of segments/10ms samples =	28.2%

Test Detail of Speaker DS, No Syntax, on Test Data recorded with Training Data.

```
1   UTT="STORE NEGATE ALPHA INTO EPSILON"
    REC="STORE IN EIGHT ALPHA INTO EPSILON"                    CORRECT=4/5
2   UTT="PUT ABSOLUTE BETA IN DELTA"
    REC="PUT ABSOLUTE BETA IN DELTA"                           CORRECT=5/5
3   UTT="ALPHA GETS FACT GAMMA"
    REC="ALPHA GETS FACT GAMMA"                                CORRECT=4/4
4   UTT="BETA BECOMES OCTAL ONE ZERO"
    REC="BETA BECOMES WHAT OCTAL ONE IN ZERO"                  CORRECT=4/5
5   UTT="GAMMA GETS DECIMAL TWO THREE"
    REC="GAMMA GETS DECIMAL TWO THREE"                         CORRECT=5/5
6   UTT="DELTA BECOMES FOUR PLUS FIVE"
    REC="WHAT BECOMES FOUR PLUS FIVE"                          CORRECT=4/5
7   UTT="EPSILON GETS SIX EIGHT MINUS DELTA"
    REC="EPSILON GETS SIX EIGHT MINUS DELTA"                   CORRECT=6/6
8   UTT="WHAT IS SEVEN NINE TIMES EPSILON"
    REC="WHAT IS SEVEN NINE TIMES IN EPSILON"                  CORRECT=5/6
9   UTT="SHOW ONE ZERO TWO DIVIDE THREE FOUR"
    REC="SHOW ONE ZERO TWO DIVIDE THREE FOUR"                  CORRECT=7/7
10  UTT="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"
    REC="STORE FIVE SIX MOD SEVEN EIGHT INTO WHAT ALPHA"       CORRECT=7/8
11  UTT="PUT NINE POWER TWO ONE IN BETA"
    REC="PUT NINE POWER TWO ONE IN BETA"                       CORRECT=7/7
12  UTT="ALPHA BECOMES THREE ZERO MAX GAMMA"
    REC="ALPHA BECOMES THREE ZERO MAX GAMMA"                   CORRECT=6/6
13  UTT="BETA GETS DECIMAL FOUR SIX MIN DELTA"
    REC="EIGHT IN GETS DECIMAL FOUR SIX MIN DELTA"             CORRECT=6/7
14  UTT="GAMMA BECOMES NEGATE EPSILON"
    REC="GAMMA BECOMES NEGATE EPSILON"                         CORRECT=4/4
15  UTT="DELTA GETS ABSOLUTE ALPHA"
    REC="DELTA GETS ABSOLUTE ALPHA"                            CORRECT=4/4
16  UTT="EPSILON BECOMES FACT BETA"
    REC="EPSILON BECOMES FACT BETA"                            CORRECT=4/4
17  UTT="WHAT IS SEVEN PLUS EIGHT"
    REC="WHAT GETS SEVEN PLUS EIGHT"                           CORRECT=4/5
18  UTT="SHOW NINE MINUS FIVE"
    REC="SHOW NINE MINUS FIVE"                                 CORRECT=4/4
19  UTT="STORE TWO ZERO TIMES THREE INTO GAMMA"
    REC="STORE TWO ZERO TIMES FOUR EIGHT INTO GAMMA"           CORRECT=6/7
20  UTT="PUT FOUR DIVIDE DELTA IN EPSILON"
    REC="PUT FOUR TWO NINE DELTA IN EPSILON"                   CORRECT=5/6
```

Test Summary of Speaker RG, No Syntax, on Test Data recorded with Training Data.

UTT	LNG	WORDS	#COR	%COR	R-TIME	#STATES	SEGS
1	2.1	5	5	100.0	9.9	52	62
2	1.8	5	4	80.0	10.2	56	50
3	1.5	4	3	75.0	13.5	78	51
4	1.9	5	5	100.0	9.7	43	55
5	1.9	5	5	100.0	10.8	63	55
6	1.8	5	4	80.0	9.3	46	53
7	2.4	6	6	100.0	10.3	58	68
8	2.2	6	6	100.0	8.7	45	59
9	2.7	7	7	100.0	9.7	47	82
10	2.1	7	7	100.0	9.0	49	56
11	2.3	6	6	100.0	10.2	45	69
12	2.6	7	7	100.0	11.8	56	82
13	1.8	4	3	75.0	13.0	61	60
14	1.8	4	4	100.0	12.8	60	63
15	1.8	4	3	75.0	10.3	56	53
16	1.5	5	5	100.0	9.2	55	42
17	1.4	4	3	75.0	5.4	22	33
18	2.6	7	7	100.0	10.2	55	85
19	2.2	6	5	83.3	10.3	62	69

LNG	= Utterance length in seconds
WORDS	= Number of words in utterance
#COR	= Number of words correctly recognized
%COR	= % of words correctly recognized
R-TIME	= Number of times real time needed for recognition ADD 8.4 for ACS and LPC calculations.
#STATES	= Average number of states checked at each segment
SEGS	= Number of segments produced (LNGx100 = number of 10ms. time samples)

Total words =	102
Correct words =	95 = 93.1%
Correct utterances =	12 = 63.2%
Average search time =	10.2 times real time
Including ACS and LPC calculation =	18.6
Average number of states searched =	53
Ratio of segments/10ms samples =	30.2%

Test Detail of Speaker RG, No Syntax, on Test Data recorded with Training Data.

```
1    UTT="STORE NEGATE ALPHA INTO EPSILON"
     REC="STORE NEGATE ALPHA INTO EPSILON"                 CORRECT=5/5
2    UTT="PUT ABSOLUTE BETA IN DELTA"
     REC="MOD ABSOLUTE BETA IN DELTA"                      CORRECT=4/5
3    UTT="ALPHA GETS FACT GAMMA"
     REC="ALPHA GETS FIVE GAMMA"                           CORRECT=3/4
4    UTT="BETA BECOMES OCTAL ONE ZERO"
     REC="BETA BECOMES OCTAL ONE ZERO"                     CORRECT=5/5
5    UTT="GAMMA GETS DECIMAL TWO THREE"
     REC="GAMMA GETS DECIMAL TWO THREE"                    CORRECT=5/5
6    UTT="DELTA BECOMES FOUR PLUS FIVE"
     REC="DELTA BECOMES FOUR PLUS MOD IN"                  CORRECT=4/5
7    UTT="EPSILON GETS SIX EIGHT MINUS DELTA"
     REC="EPSILON GETS SIX EIGHT MINUS DELTA"              CORRECT=6/6
8    UTT="WHAT IS SEVEN NINE TIMES EPSILON"
     REC="WHAT IS SEVEN NINE TIMES EPSILON"                CORRECT=6/6
9    UTT="SHOW ONE ZERO TWO DIVIDE THREE FOUR"
     REC="SHOW ONE ZERO TWO DIVIDE THREE FOUR"             CORRECT=7/7
10   UTT="PUT NINE POWER TWO ONE IN BETA"
     REC="PUT NINE POWER TWO ONE IN BETA"                  CORRECT=7/7
11   UTT="ALPHA BECOMES THREE ZERO MAX GAMMA"
     REC="ALPHA BECOMES THREE ZERO MAX GAMMA"              CORRECT=6/6
12   UTT="BETA GETS DECIMAL FOUR SIX MIN DELTA"
     REC="BETA GETS DECIMAL FOUR SIX MIN DELTA"            CORRECT=7/7
13   UTT="GAMMA BECOMES NEGATE EPSILON"
     REC="GAMMA BECOMES MIN BETA EPSILON"                  CORRECT=3/4
14   UTT="DELTA GETS ABSOLUTE ALPHA"
     REC="DELTA GETS ABSOLUTE ALPHA"                       CORRECT=4/4
15   UTT="EPSILON BECOMES FACT BETA"
     REC="EPSILON BECOMES FACT ZERO"                       CORRECT=3/4
16   UTT="WHAT IS SEVEN PLUS EIGHT"
     REC="WHAT IS SEVEN PLUS EIGHT"                        CORRECT=5/5
17   UTT="SHOW NINE MINUS FIVE"
     REC="SHOW NINE MINUS MOD"                             CORRECT=3/4
18   UTT="STORE TWO ZERO TIMES THREE INTO GAMMA"
     REC="STORE TWO ZERO TIMES THREE INTO GAMMA"           CORRECT=7/7
19   UTT="PUT FOUR DIVIDE DELTA IN EPSILON"
     REC="WHAT FOUR DIVIDE DELTA IN EPSILON"               CORRECT=5/6
```

Test Summary of Speaker BL, With Syntax, on Test Data recorded with Training Data.

UTT	LNG	WORDS	#COR	%COR	R-TIME	#STATES	SEGS
1	2.0	5	5	100.0	3.7	15	62
2	1.6	5	4	80.0	4.2	23	49
3	1.5	4	4	100.0	4.3	22	44
4	2.0	5	5	100.0	4.7	24	63
5	1.7	5	5	100.0	4.5	21	58
6	1.9	5	5	100.0	4.3	20	60
7	2.2	6	6	100.0	5.6	34	74
8	2.0	6	6	100.0	5.5	36	62
9	2.5	7	7	100.0	5.0	28	72
10	2.7	8	6	75.0	4.3	22	76
11	2.3	7	7	100.0	3.9	19	63
12	2.3	6	6	100.0	3.9	17	70
13	2.6	7	7	100.0	4.8	26	86
14	2.0	4	4	100.0	4.0	23	61
15	1.8	4	4	100.0	5.0	30	62
16	2.0	4	4	100.0	3.9	22	54
17	1.7	5	5	100.0	4.3	24	45
18	1.5	4	4	100.0	3.2	15	36
19	2.4	7	7	100.0	5.0	25	80
20	2.2	6	6	100.0	4.4	25	67

LNG	= Utterance length in seconds
WORDS	= Number of words in utterance
#COR	= Number of words correctly recognized
%COR	= % of words correctly recognized
R-TIME	= Number of times real time needed for recognition ADD 8.4 for ACS and LPC calculations.
#STATES	= Average number of states checked at each segment
SEGS	= Number of segments produced (LNGx100 = number of 10ms. time samples)

Total words =	110
Correct words =	107 = 97.3%
Correct utterances =	18 = 90.0%
Average search time =	4.5 times real time
Including ACS and LPC calculation =	12.9
Average number of states searched =	23
Ratio of segments/10ms samples =	30.6%

**Test Detail of Speaker BL, With Syntax,
on Test Data recorded with Training Data.**

```
1   UTT="STORE NEGATE ALPHA INTO EPSILON"
    REC="STORE NEGATE ALPHA INTO EPSILON"                CORRECT=5/5
2   UTT="PUT ABSOLUTE BETA IN DELTA"
    REC="PUT ABSOLUTE EIGHT IN DELTA"                    CORRECT=4/5
3   UTT="ALPHA GETS FACT GAMMA"
    REC="ALPHA GETS FACT GAMMA"                          CORRECT=4/4
4   UTT="BETA BECOMES OCTAL ONE ZERO"
    REC="BETA BECOMES OCTAL ONE ZERO"                    CORRECT=5/5
5   UTT="GAMMA GETS DECIMAL TWO THREE"
    REC="GAMMA GETS DECIMAL TWO THREE"                   CORRECT=5/5
6   UTT="DELTA BECOMES FOUR PLUS FIVE"
    REC="DELTA BECOMES FOUR PLUS FIVE"                   CORRECT=5/5
7   UTT="EPSILON GETS SIX EIGHT MINUS DELTA"
    REC="EPSILON GETS SIX EIGHT MINUS DELTA"             CORRECT=6/6
8   UTT="WHAT IS SEVEN NINE TIMES EPSILON"
    REC="WHAT IS SEVEN NINE TIMES EPSILON"               CORRECT=6/6
9   UTT="SHOW ONE ZERO TWO DIVIDE THREE FOUR"
    REC="SHOW ONE ZERO TWO DIVIDE THREE FOUR"            CORRECT=7/7
10  UTT="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"
    REC="STORE FIVE SIX NINE SEVEN INTO ALPHA"           CORRECT=6/8
11  UTT="PUT NINE POWER TWO ONE IN BETA"
    REC="PUT NINE POWER TWO ONE IN BETA"                 CORRECT=7/7
12  UTT="ALPHA BECOMES THREE ZERO MAX GAMMA"
    REC="ALPHA BECOMES THREE ZERO MAX GAMMA"             CORRECT=6/6
13  UTT="BETA GETS DECIMAL FOUR SIX MIN DELTA"
    REC="BETA GETS DECIMAL FOUR SIX MIN DELTA"           CORRECT=7/7
14  UTT="GAMMA BECOMES NEGATE EPSILON"
    REC="GAMMA BECOMES NEGATE EPSILON"                   CORRECT=4/4
15  UTT="DELTA GETS ABSOLUTE ALPHA"
    REC="DELTA GETS ABSOLUTE ALPHA"                      CORRECT=4/4
16  UTT="EPSILON BECOMES FACT BETA"
    REC="EPSILON BECOMES FACT BETA"                      CORRECT=4/4
17  UTT="WHAT IS SEVEN PLUS EIGHT"
    REC="WHAT IS SEVEN PLUS EIGHT"                       CORRECT=5/5
18  UTT="SHOW NINE MINUS FIVE"
    REC="SHOW NINE MINUS FIVE"                           CORRECT=4/4
19  UTT="STORE TWO ZERO TIMES THREE INTO GAMMA"
    REC="STORE TWO ZERO TIMES THREE INTO GAMMA"          CORRECT=7/7
20  UTT="PUT FOUR DIVIDE DELTA IN EPSILON"
    REC="PUT FOUR DIVIDE DELTA IN EPSILON"               CORRECT=6/6
```

Test Summary of Speaker KP, With Syntax, on Test Data recorded with Training Data.

UTT	LNG	WORDS	#COR	%COR	R-TIME	#STATES	SEGS
1	2.1	5	5	100.0	3.7	16	55
2	1.9	5	5	100.0	5.3	32	52
3	1.7	4	4	100.0	4.7	26	46
4	2.1	5	5	100.0	5.5	30	62
5	1.9	5	5	100.0	5.5	34	53
6	2.0	5	4	80.0	4.3	22	53
7	2.4	6	6	100.0	5.4	41	69
8	2.4	6	6	100.0	4.3	32	58
9	2.7	7	6	85.7	6.1	50	73
10	3.2	8	8	100.0	4.1	28	85
11	2.4	7	7	100.0	3.9	28	61
12	2.5	6	6	100.0	3.7	20	64
13	2.6	7	6	85.7	5.2	36	82
14	2.1	4	4	100.0	3.7	23	56
15	1.8	4	4	100.0	4.4	30	52
16	2.0	4	4	100.0	3.4	27	44
17	1.7	5	5	100.0	4.8	38	41
18	1.7	4	2	50.0	3.1	17	31
19	2.8	7	7	100.0	4.6	31	74
20	2.4	6	6	100.0	4.6	29	68

LNG	= Utterance length in seconds
WORDS	= Number of words in utterance
#COR	= Number of words correctly recognized
%COR	= % of words correctly recognized
R-TIME	= Number of times real time needed for recognition ADD 8.4 for ACS and LPC calculations.
#STATES	= Average number of states checked at each segment
SEGS	= Number of segments produced (LNGx100 = number of 10ms. time samples)

Total words =	110
Correct words =	105 = 95.5%
Correct utterances =	16 = 80.0%
Average search time =	4.5 times real time
Including ACS and LPC calculation =	12.9
Average number of states searched =	29
Ratio of segments/10ms samples =	26.7%

Test Detail of Speaker KP, With Syntax, on Test Data recorded with Training Data.

```
1   UTT="STORE NEGATE ALPHA INTO EPSILON"
    REC="STORE NEGATE ALPHA INTO EPSILON"                 CORRECT=5/5
2   UTT="PUT ABSOLUTE BETA IN DELTA"
    REC="PUT ABSOLUTE BETA IN DELTA"                      CORRECT=5/5
3   UTT="ALPHA GETS FACT GAMMA"
    REC="ALPHA GETS FACT GAMMA"                           CORRECT=4/4
4   UTT="BETA BECOMES OCTAL ONE ZERO"
    REC="BETA BECOMES OCTAL ONE ZERO"                     CORRECT=5/5
5   UTT="GAMMA GETS DECIMAL TWO THREE"
    REC="GAMMA GETS DECIMAL TWO THREE"                    CORRECT=5/5
6   UTT="DELTA BECOMES FOUR PLUS FIVE"
    REC="DELTA BECOMES FOUR PLUS ONE"                     CORRECT=4/5
7   UTT="EPSILON GETS SIX EIGHT MINUS DELTA"
    REC="EPSILON GETS SIX EIGHT MINUS DELTA"              CORRECT=6/6
8   UTT="WHAT IS SEVEN NINE TIMES EPSILON"
    REC="WHAT IS SEVEN NINE TIMES EPSILON"                CORRECT=6/6
9   UTT="SHOW ONE ZERO TWO DIVIDE THREE FOUR"
    REC="SHOW ONE ZERO TWO TWO FIVE THREE FOUR"           CORRECT=6/7
10  UTT="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"
    REC="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"       CORRECT=8/8
11  UTT="PUT NINE POWER TWO ONE IN BETA"
    REC="PUT NINE POWER TWO ONE IN BETA"                  CORRECT=7/7
12  UTT="ALPHA BECOMES THREE ZERO MAX GAMMA"
    REC="ALPHA BECOMES THREE ZERO MAX GAMMA"              CORRECT=6/6
13  UTT="BETA GETS DECIMAL FOUR SIX MIN DELTA"
    REC="BETA GETS DECIMAL FOUR EIGHT SIX MIN DELTA"      CORRECT=6/7
14  UTT="GAMMA BECOMES NEGATE EPSILON"
    REC="GAMMA BECOMES NEGATE EPSILON"                    CORRECT=4/4
15  UTT="DELTA GETS ABSOLUTE ALPHA"
    REC="DELTA GETS ABSOLUTE ALPHA"                       CORRECT=4/4
16  UTT="EPSILON BECOMES FACT BETA"
    REC="EPSILON BECOMES FACT BETA"                       CORRECT=4/4
17  UTT="WHAT IS SEVEN PLUS EIGHT"
    REC="WHAT IS SEVEN PLUS EIGHT"                        CORRECT=5/5
18  UTT="SHOW NINE MINUS FIVE"
    REC="SHOW NINE MOD EPSILON"                           CORRECT=2/4
19  UTT="STORE TWO ZERO TIMES THREE INTO GAMMA"
    REC="STORE TWO ZERO TIMES THREE INTO GAMMA"           CORRECT=7/7
20  UTT="PUT FOUR DIVIDE DELTA IN EPSILON"
    REC="PUT FOUR DIVIDE DELTA IN EPSILON"                CORRECT=6/6
```

Test Summary of Speaker DS, With Syntax, on Test Data recorded with Training Data.

UTT	LNG	WORDS	#COR	7COR	R-TIME	#STATES	SEGS
1	2.0	5	5	100.0	4.0	23	61
2	2.0	5	5	100.0	3.9	28	57
3	1.8	4	4	100.0	4.0	28	55
4	2.4	5	4	80.0	4.1	25	68
5	2.2	5	5	100.0	4.3	27	60
6	2.2	5	5	100.0	4.0	21	60
7	2.6	6	6	100.0	4.4	30	71
8	2.4	6	6	100.0	5.0	36	65
9	3.3	7	7	100.0	7.0	56	89
10	3.5	8	8	100.0	4.1	25	93
11	2.6	7	7	100.0	4.8	32	70
12	2.9	6	6	100.0	4.9	26	87
13	3.1	7	7	100.0	4.6	26	91
14	2.1	4	4	100.0	3.2	15	62
15	1.9	4	4	100.0	5.2	37	57
16	2.4	4	4	100.0	4.3	31	62
17	1.9	5	5	100.0	5.1	50	48
18	1.9	4	4	100.0	3.4	19	42
19	3.0	7	6	85.7	5.2	33	94
20	2.5	6	6	100.0	5.5	42	76

LNG	= Utterance length in seconds
WORDS	= Number of words in utterance
#COR	= Number of words correctly recognized
7COR	= 7 of words correctly recognized
R-TIME	= Number of times real time needed for recognition ADD 8.4 for ACS and LPC calculations.
#STATES	= Average number of states checked at each segment
SEGS	= Number of segments produced (LNGx100 = number of 10ms. time samples)

Total words =	110
Correct words =	108 = 98.27
Correct utterances =	18 = 90.07
Average search time =	4.6 times real time
Including ACS and LPC calculation =	13.0
Average number of states searched =	30
Ratio of segments/10ms samples =	28.27

Test Detail of Speaker DS, With Syntax, on Test Data recorded with Training Data.

```
1   UTT="STORE NEGATE ALPHA INTO EPSILON"
    REC="STORE NEGATE ALPHA INTO EPSILON"                  CORRECT=5/5
2   UTT="PUT ABSOLUTE BETA IN DELTA"
    REC="PUT ABSOLUTE BETA IN DELTA"                       CORRECT=5/5
3   UTT="ALPHA GETS FACT GAMMA"
    REC="ALPHA GETS FACT GAMMA"                            CORRECT=4/4
4   UTT="BETA BECOMES OCTAL ONE ZERO"
    REC="BETA BECOMES OCTAL ONE MIN ZERO"                  CORRECT=4/5
5   UTT="GAMMA GETS DECIMAL TWO THREE"
    REC="GAMMA GETS DECIMAL TWO THREE"                     CORRECT=5/5
6   UTT="DELTA BECOMES FOUR PLUS FIVE"
    REC="DELTA BECOMES FOUR PLUS FIVE"                     CORRECT=5/5
7   UTT="EPSILON GETS SIX EIGHT MINUS DELTA"
    REC="EPSILON GETS SIX EIGHT MINUS DELTA"               CORRECT=6/6
8   UTT="WHAT IS SEVEN NINE TIMES EPSILON"
    REC="WHAT IS SEVEN NINE TIMES EPSILON"                 CORRECT=6/6
9   UTT="SHOW ONE ZERO TWO DIVIDE THREE FOUR"
    REC="SHOW ONE ZERO TWO DIVIDE THREE FOUR"              CORRECT=7/7
10  UTT="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"
    REC="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"        CORRECT=8/8
11  UTT="PUT NINE POWER TWO ONE IN BETA"
    REC="PUT NINE POWER TWO ONE IN BETA"                   CORRECT=7/7
12  UTT="ALPHA BECOMES THREE ZERO MAX GAMMA"
    REC="ALPHA BECOMES THREE ZERO MAX GAMMA"               CORRECT=6/6
13  UTT="BETA GETS DECIMAL FOUR SIX MIN DELTA"
    REC="BETA GETS DECIMAL FOUR SIX MIN DELTA"             CORRECT=7/7
14  UTT="GAMMA BECOMES NEGATE EPSILON"
    REC="GAMMA BECOMES NEGATE EPSILON"                     CORRECT=4/4
15  UTT="DELTA GETS ABSOLUTE ALPHA"
    REC="DELTA GETS ABSOLUTE ALPHA"                        CORRECT=4/4
16  UTT="EPSILON BECOMES FACT BETA"
    REC="EPSILON BECOMES FACT BETA"                        CORRECT=4/4
17  UTT="WHAT IS SEVEN PLUS EIGHT"
    REC="WHAT IS SEVEN PLUS EIGHT"                         CORRECT=5/5
18  UTT="SHOW NINE MINUS FIVE"
    REC="SHOW NINE MINUS FIVE"                             CORRECT=4/4
19  UTT="STORE TWO ZERO TIMES THREE INTO GAMMA"
    REC="STORE TWO ZERO TIMES FOUR EIGHT INTO GAMMA"       CORRECT=6/7
20  UTT="PUT FOUR DIVIDE DELTA IN EPSILON"
    REC="PUT FOUR DIVIDE DELTA IN EPSILON"                 CORRECT=6/6
```

Test Summary of Speaker RG, With Syntax, on Test Data recorded with Training Data.

UTT	LNG	WORDS	#COR	%COR	R-TIME	#STATES	SEGS
1	2.1	5	5	100.0	3.6	13	62
2	1.8	5	5	100.0	4.9	26	50
3	1.5	4	4	100.0	6.7	48	51
4	1.9	5	5	100.0	4.7	22	55
5	1.9	5	5	100.0	5.3	29	55
6	1.8	5	5	100.0	4.5	23	53
7	2.4	6	6	100.0	5.0	29	68
8	2.2	6	6	100.0	5.3	30	59
9	2.7	7	7	100.0	5.7	27	82
10	2.1	7	7	100.0	4.2	21	56
11	2.3	6	6	100.0	5.1	23	69
12	2.6	7	7	100.0	5.5	26	82
13	1.8	4	4	100.0	5.8	31	60
14	1.8	4	4	100.0	5.3	24	63
15	1.8	4	3	75.0	5.4	29	53
16	1.5	5	5	100.0	5.6	32	42
17	1.4	4	4	100.0	3.6	13	33
18	2.6	7	7	100.0	5.7	26	85
19	2.2	6	6	100.0	8.4	61	69

LNG	= Utterance length in seconds
WORDS	= Number of words in utterance
#COR	= Number of words correctly recognized
%COR	= % of words correctly recognized
R-TIME	= Number of times real time needed for recognition ADD 8.4 for ACS and LPC calculations.
#STATES	= Average number of states checked at each segment
SEGS	= Number of segments produced (LNGx100 = number of 10ms. time samples)

Total words =	102
Correct words =	101 = 99.0%
Correct utterances =	18 = 94.7%
Average search time =	5.3 times real time
Including ACS and LPC calculation =	13.7
Average number of states searched =	28
Ratio of segments/10ms samples =	30.2%

Test Detail of Speaker RG, With Syntax, on Test Data recorded with Training Data.

```
1    UTT="STORE NEGATE ALPHA INTO EPSILON"
     REC="STORE NEGATE ALPHA INTO EPSILON"                CORRECT=5/5
2    UTT="PUT ABSOLUTE BETA IN DELTA"
     REC="PUT ABSOLUTE BETA IN DELTA"                     CORRECT=5/5
3    UTT="ALPHA GETS FACT GAMMA"
     REC="ALPHA GETS FACT GAMMA"                          CORRECT=4/4
4    UTT="BETA BECOMES OCTAL ONE ZERO"
     REC="BETA BECOMES OCTAL ONE ZERO"                    CORRECT=5/5
5    UTT="GAMMA GETS DECIMAL TWO THREE"
     REC="GAMMA GETS DECIMAL TWO THREE"                   CORRECT=5/5
6    UTT="DELTA BECOMES FOUR PLUS FIVE"
     REC="DELTA BECOMES FOUR PLUS FIVE"                   CORRECT=5/5
7    UTT="EPSILON GETS SIX EIGHT MINUS DELTA"
     REC="EPSILON GETS SIX EIGHT MINUS DELTA"             CORRECT=6/6
8    UTT="WHAT IS SEVEN NINE TIMES EPSILON"
     REC="WHAT IS SEVEN NINE TIMES EPSILON"               CORRECT=6/6
9    UTT="SHOW ONE ZERO TWO DIVIDE THREE FOUR"
     REC="SHOW ONE ZERO TWO DIVIDE THREE FOUR"            CORRECT=7/7
10   UTT="PUT NINE POWER TWO ONE IN BETA"
     REC="PUT NINE POWER TWO ONE IN BETA"                 CORRECT=7/7
11   UTT="ALPHA BECOMES THREE ZERO MAX GAMMA"
     REC="ALPHA BECOMES THREE ZERO MAX GAMMA"             CORRECT=6/6
12   UTT="BETA GETS DECIMAL FOUR SIX MIN DELTA"
     REC="BETA GETS DECIMAL FOUR SIX MIN DELTA"           CORRECT=7/7
13   UTT="GAMMA BECOMES NEGATE EPSILON"
     REC="GAMMA BECOMES NEGATE EPSILON"                   CORRECT=4/4
14   UTT="DELTA GETS ABSOLUTE ALPHA"
     REC="DELTA GETS ABSOLUTE ALPHA"                      CORRECT=4/4
15   UTT="EPSILON BECOMES FACT BETA"
     REC="EPSILON BECOMES FACT ZERO"                      CORRECT=3/4
16   UTT="WHAT IS SEVEN PLUS EIGHT"
     REC="WHAT IS SEVEN PLUS EIGHT"                       CORRECT=5/5
17   UTT="SHOW NINE MINUS FIVE"
     REC="SHOW NINE MINUS FIVE"                           CORRECT=4/4
18   UTT="STORE TWO ZERO TIMES THREE INTO GAMMA"
     REC="STORE TWO ZERO TIMES THREE INTO GAMMA"          CORRECT=7/7
19   UTT="PUT FOUR DIVIDE DELTA IN EPSILON"
     REC="PUT FOUR DIVIDE DELTA IN EPSILON"               CORRECT=6/6
```

Test Summary of Speaker BL, No Syntax,
on Test Data recorded 5 Months after Training Data.

UTT	LNG	WORDS	#COR	%COR	R-TIME	#STATES	SEGS
1	2.1	5	4	80.0	9.4	55	63
2	1.8	5	5	100.0	9.9	54	54
3	1.6	4	4	100.0	9.9	62	46
4	2.0	5	5	100.0	8.3	40	59
5	1.8	5	5	100.0	10.5	56	58
6	1.9	5	4	80.0	10.1	53	64
7	2.3	6	6	100.0	9.1	51	68
8	2.1	6	6	100.0	9.2	53	65
9	2.6	7	6	85.7	8.7	55	72
10	2.7	8	6	75.0	9.5	50	90
11	2.0	7	5	71.4	9.4	51	61
12	2.3	6	6	100.0	9.4	45	77
13	2.5	7	6	85.7	9.3	50	77
14	1.9	4	3	75.0	10.3	62	62
15	1.7	4	4	100.0	9.9	47	59
16	1.9	4	4	100.0	9.5	56	59
17	1.6	5	4	80.0	7.3	38	42
18	1.6	4	4	100.0	6.3	30	40
19	2.7	7	7	100.0	9.5	52	86
20	2.3	6	5	83.3	10.0	57	71

LNG	= Utterance length in seconds
WORDS	= Number of words in utterance
#COR	= Number of words correctly recognized
%COR	= % of words correctly recognized
R-TIME	= Number of times real time needed for recognition ADD 8.4 for ACS and LPC calculations.
#STATES	= Average number of states checked at each segment
SEGS	= Number of segments produced (LNGx100 = number of 10ms. time samples)

Total words =	110
Correct words =	99 = 90.0%
Correct utterances =	11 = 55.0%
Average search time =	9.3 times real time
Including ACS and LPC calculation =	17.7 times real time
Average number of states searched =	50
Ratio of segments/10ms samples =	30.9%

Test Detail of Speaker BL, No Syntax, on Test Data recorded 5 Months after Training Data.

```
1    UTT="STORE NEGATE ALPHA INTO EPSILON"
     REC="STORE NEGATE DELTA INTO EPSILON"                  CORRECT=4/5
2    UTT="PUT ABSOLUTE BETA IN DELTA"
     REC="PUT ABSOLUTE BETA IN DELTA"                       CORRECT=5/5
3    UTT="ALPHA GETS FACT GAMMA"
     REC="ALPHA GETS FACT GAMMA"                            CORRECT=4/4
4    UTT="BETA BECOMES OCTAL ONE ZERO"
     REC="BETA BECOMES OCTAL ONE ZERO"                      CORRECT=5/5
5    UTT="GAMMA GETS DECIMAL TWO THREE"
     REC="GAMMA GETS DECIMAL TWO THREE"                     CORRECT=5/5
6    UTT="DELTA BECOMES FOUR PLUS FIVE"
     REC="DELTA BECOMES FOUR PLUS NINE"                     CORRECT=4/5
7    UTT="EPSILON GETS SIX EIGHT MINUS DELTA"
     REC="EPSILON GETS SIX EIGHT MINUS DELTA"               CORRECT=6/6
8    UTT="WHAT IS SEVEN NINE TIMES EPSILON"
     REC="WHAT IS SEVEN NINE TIMES EPSILON"                 CORRECT=6/6
9    UTT="SHOW ONE ZERO TWO DIVIDE THREE FOUR"
     REC="SHOW ONE ZERO TWO DIVIDE THREE WHAT"              CORRECT=6/7
10   UTT="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"
     REC="STORE FIVE SIX NINE SEVEN EIGHT MIN PUT ALPHA"    CORRECT=6/8
11   UTT="PUT NINE POWER TWO ONE IN BETA"
     REC="MOD NINE POWER TWO ONE MIN BETA"                  CORRECT=5/7
12   UTT="ALPHA BECOMES THREE ZERO MAX GAMMA"
     REC="ALPHA BECOMES THREE ZERO MAX GAMMA"               CORRECT=6/6
13   UTT="BETA GETS DECIMAL FOUR SIX MIN DELTA"
     REC="EIGHT GETS DECIMAL FOUR SIX MIN DELTA"            CORRECT=6/7
14   UTT="GAMMA BECOMES NEGATE EPSILON"
     REC="GAMMA BECOMES MIN BETA EPSILON"                   CORRECT=3/4
15   UTT="DELTA GETS ABSOLUTE ALPHA"
     REC="DELTA GETS ABSOLUTE ALPHA"                        CORRECT=4/4
16   UTT="EPSILON BECOMES FACT BETA"
     REC="EPSILON BECOMES FACT BETA"                        CORRECT=4/4
17   UTT="WHAT IS SEVEN PLUS EIGHT"
     REC="WHAT IN SEVEN PLUS EIGHT"                         CORRECT=4/5
18   UTT="SHOW NINE MINUS FIVE"
     REC="SHOW NINE MINUS FIVE"                             CORRECT=4/4
19   UTT="STORE TWO ZERO TIMES THREE INTO GAMMA"
     REC="STORE TWO ZERO TIMES THREE INTO GAMMA"            CORRECT=7/7
20   UTT="PUT FOUR DIVIDE DELTA IN EPSILON"
     REC="PUT FOUR DIVIDE DELTA IN MIN EPSILON"             CORRECT=5/6
```

Test Summary of Speaker KP, No Syntax,
on Test Data recorded 5 Months after Training Data.

UTT	LNG	WORDS	#COR	%COR	R-TIME	#STATES	SEGS
1	2.0	5	4	80.0	9.5	72	54
2	1.8	5	5	100.0	12.8	100	57
3	1.5	4	4	100.0	9.8	72	39
4	2.0	5	4	80.0	9.5	68	55
5	1.7	5	3	60.0	10.0	66	49
6	1.9	5	3	60.0	7.8	53	43
7	2.3	6	3	50.0	12.9	101	71
8	2.0	6	5	83.3	10.6	78	54
9	2.6	7	5	71.4	8.6	61	63
10	2.8	8	8	100.0	9.7	69	77
11	2.6	7	6	85.7	7.6	55	55
12	2.3	6	5	83.3	8.4	51	63
13	2.6	7	4	57.1	9.7	81	67
14	2.1	4	4	100.0	8.1	54	53
15	1.9	4	4	100.0	10.2	80	52
16	1.9	4	4	100.0	7.8	59	46
17	1.6	5	2	40.0	10.0	87	42
18	1.6	4	2	50.0	6.7	46	35
19	2.7	7	6	85.7	8.9	69	70
20	2.2	6	4	66.7	9.5	74	59

LNG	= Utterance length in seconds
WORDS	= Number of words in utterance
#COR	= Number of words correctly recognized
%COR	= % of words correctly recognized
R-TIME	= Number of times real time needed for recognition ADD 8.4 for ACS and LPC calculations.
#STATES	= Average number of states checked at each segment
SEGS	= Number of segments produced (LNGx100 = number of 10ms. time samples)

Total words =	110
Correct words =	85 = 77.3%
Correct utterances =	6 = 30.0%
Average search time =	9.4 times real time
Including ACS and LPC calculation =	17.8 times real time
Average number of states searched =	69
Ratio of segments/10ms samples =	26.2%

**Test Detail of Speaker KP, No Syntax,
on Test Data recorded 5 Months after Training Data.**

```
1   UTT="STORE NEGATE ALPHA INTO EPSILON"
    REC="STORE MIN BETA ALPHA INTO EPSILON"                  CORRECT=4/5
2   UTT="PUT ABSOLUTE BETA IN DELTA"
    REC="PUT ABSOLUTE BETA IN DELTA"                         CORRECT=5/5
3   UTT="ALPHA GETS FACT GAMMA"
    REC="ALPHA GETS FACT GAMMA"                              CORRECT=4/4
4   UTT="BETA BECOMES OCTAL ONE ZERO"
    REC="EIGHT TWO BECOMES OCTAL ONE ZERO"                   CORRECT=4/5
5   UTT="GAMMA GETS DECIMAL TWO THREE"
    REC="GAMMA BECOMES ONE TWO THREE"                        CORRECT=3/5
6   UTT="DELTA BECOMES FOUR PLUS FIVE"
    REC="DELTA BECOMES FOUR MOD NINE"                        CORRECT=3/5
7   UTT="EPSILON GETS SIX EIGHT MINUS DELTA"
    REC="EPSILON MIN GETS BETA EIGHT NINE MAX DELTA"         CORRECT=3/6
8   UTT="WHAT IS SEVEN NINE TIMES EPSILON"
    REC="MOD IS SEVEN NINE TIMES EPSILON"                    CORRECT=5/6
9   UTT="SHOW ONE ZERO TWO DIVIDE THREE FOUR"
    REC="SHOW ONE ZERO TWO TWO NINE THREE ONE"               CORRECT=5/7
10  UTT="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"
    REC="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"          CORRECT=8/8
11  UTT="PUT NINE POWER TWO ONE IN BETA"
    REC="PUT NINE POWER TWO IN ONE IN BETA"                  CORRECT=6/7
12  UTT="ALPHA BECOMES THREE ZERO MAX GAMMA"
    REC="OCTAL BECOMES THREE ZERO MAX GAMMA"                 CORRECT=5/6
13  UTT="BETA GETS DECIMAL FOUR SIX MIN DELTA"
    REC="BETA GETS DECIMAL ONE FACT IN DELTA"                CORRECT=4/7
14  UTT="GAMMA BECOMES NEGATE EPSILON"
    REC="GAMMA BECOMES NEGATE EPSILON"                       CORRECT=4/4
15  UTT="DELTA GETS ABSOLUTE ALPHA"
    REC="DELTA GETS ABSOLUTE ALPHA"                          CORRECT=4/4
16  UTT="EPSILON BECOMES FACT BETA"
    REC="EPSILON BECOMES FACT BETA"                          CORRECT=4/4
17  UTT="WHAT IS SEVEN PLUS EIGHT"
    REC="WHAT IS WHAT IN MOD IN"                             CORRECT=2/5
18  UTT="SHOW NINE MINUS FIVE"
    REC="SHOW NINE MIN ONE"                                  CORRECT=2/4
19  UTT="STORE TWO ZERO TIMES THREE INTO GAMMA"
    REC="STORE TWO ZERO TWO ONE THREE INTO GAMMA"            CORRECT=6/7
20  UTT="PUT FOUR DIVIDE DELTA IN EPSILON"
    REC="IN FOUR WHAT FIVE DELTA IN EPSILON"                 CORRECT=4/6
```

Test Summary of Speaker DS, No Syntax, on Test Data recorded 5 Months after Training Data.

UTT	LNG	WORDS	#COR	%COR	R-TIME	#STATES	SEGS
1	2.1	5	3	60.0	9.5	46	61
2	1.9	5	5	100.0	11.2	65	57
3	1.8	4	3	75.0	9.0	62	47
4	2.3	5	4	80.0	9.2	51	69
5	2.2	5	4	80.0	9.8	67	61
6	2.1	5	5	100.0	7.4	43	54
7	2.6	6	6	100.0	9.0	60	75
8	2.2	6	6	100.0	9.6	45	63
9	2.8	7	6	85.7	9.4	56	82
10	3.1	8	8	100.0	8.8	54	86
11	2.4	7	6	85.7	10.1	66	72
12	2.5	6	5	83.3	8.5	48	75
13	2.7	7	6	85.7	10.7	70	79
14	2.0	4	1	25.0	11.3	68	64
15	1.9	4	4	100.0	11.2	62	62
16	2.0	4	3	75.0	10.4	68	60
17	1.7	5	4	80.0	11.0	69	49
18	1.8	4	3	75.0	7.6	47	46
19	2.7	7	4	57.1	11.6	65	86
20	2.3	6	4	66.7	10.9	69	73

LNG	= Utterance length in seconds
WORDS	= Number of words in utterance
#COR	= Number of words correctly recognized
%COR	= % of words correctly recognized
R-TIME	= Number of times real time needed for recognition ADD 8.4 for ACS and LPC calculations.
#STATES	= Average number of states checked at each segment
SEGS	= Number of segments produced (LNGx100 = number of 10ms. time samples)

Total words =	110
Correct words =	90 = 81.8%
Correct utterances =	6 = 30.0%
Average search time =	9.8 times real time
Including ACS and LPC calculation =	18.2 times real time
Average number of states searched =	59
Ratio of segments/10ms samples =	29.2%

Test Detail of Speaker DS, No Syntax,
on Test Data recorded 5 Months after Training Data.

```
1   UTT="STORE NEGATE ALPHA INTO EPSILON"
    REC="STORE IN EIGHT BETA ALPHA IN EPSILON"                  CORRECT=3/5
2   UTT="PUT ABSOLUTE BETA IN DELTA"
    REC="PUT ABSOLUTE BETA IN DELTA"                            CORRECT=5/5
3   UTT="ALPHA GETS FACT GAMMA"
    REC="MOD WHAT GETS FACT GAMMA"                              CORRECT=3/4
4   UTT="BETA BECOMES OCTAL ONE ZERO"
    REC="BETA BECOMES WHAT OCTAL ONE ZERO"                      CORRECT=4/5
5   UTT="GAMMA GETS DECIMAL TWO THREE"
    REC="GAMMA MIN DECIMAL TWO THREE"                           CORRECT=4/5
6   UTT="DELTA BECOMES FOUR PLUS FIVE"
    REC="DELTA BECOMES FOUR PLUS FIVE"                          CORRECT=5/5
7   UTT="EPSILON GETS SIX EIGHT MINUS DELTA"
    REC="EPSILON GETS SIX EIGHT MINUS DELTA"                    CORRECT=6/6
8   UTT="WHAT IS SEVEN NINE TIMES EPSILON"
    REC="WHAT IS SEVEN NINE TIMES EPSILON"                      CORRECT=6/6
9   UTT="SHOW ONE ZERO TWO DIVIDE THREE FOUR"
    REC="SHOW NINE ZERO TWO DIVIDE THREE FOUR"                  CORRECT=6/7
10  UTT="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"
    REC="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"             CORRECT=8/8
11  UTT="PUT NINE POWER TWO ONE IN BETA"
    REC="PUT NINE POWER TWO ONE MIN EIGHT BETA"                 CORRECT=6/7
12  UTT="ALPHA BECOMES THREE ZERO MAX GAMMA"
    REC="ALPHA BECOMES THREE ZERO MAX ZERO"                     CORRECT=5/6
13  UTT="BETA GETS DECIMAL FOUR SIX MIN DELTA"
    REC="IN GETS DECIMAL FOUR SIX MIN DELTA"                    CORRECT=6/7
14  UTT="GAMMA BECOMES NEGATE EPSILON"
    REC="ZERO BECOMES MIN EIGHT MAX NINE"                       CORRECT=1/4
15  UTT="DELTA GETS ABSOLUTE ALPHA"
    REC="DELTA GETS ABSOLUTE ALPHA"                             CORRECT=4/4
16  UTT="EPSILON BECOMES FACT BETA"
    REC="EPSILON BECOMES FACT IN MIN"                           CORRECT=3/4
17  UTT="WHAT IS SEVEN PLUS EIGHT"
    REC="WHAT IN SEVEN PLUS IN EIGHT"                           CORRECT=4/5
18  UTT="SHOW NINE MINUS FIVE"
    REC="SHOW MIN NINE MINUS ALPHA"                             CORRECT=3/4
19  UTT="STORE TWO ZERO TIMES THREE INTO GAMMA"
    REC="STORE TWO ZERO BECOMES STORE EIGHT IN GAMMA"           CORRECT=4/7
20  UTT="PUT FOUR DIVIDE DELTA IN EPSILON"
    REC="PUT FOUR PUT NINE DELTA IN IN EPSILON"                 CORRECT=4/6
```

Test Summary of Speaker RG, No Syntax, on Test Data recorded 5 Months after Training Data.

UTT	LNG	WORDS	*COR	%COR	R-TIME	*STATES	SEGS
1	1.9	5	4	80.0	10.7	60	56
2	1.7	5	3	60.0	11.0	62	54
3	1.5	4	4	100.0	8.5	54	38
4	1.9	5	5	100.0	9.3	43	57
5	1.9	5	4	80.0	11.3	63	58
6	1.7	5	4	80.0	10.5	50	53
7	2.4	6	4	66.7	10.4	55	76
8	2.0	6	5	83.3	8.0	49	52
9	2.7	7	6	85.7	8.2	47	77
10	3.1	8	6	75.0	9.7	61	91
11	2.1	7	6	85.7	10.2	57	65
12	2.6	6	6	100.0	8.7	44	68
13	2.7	7	6	85.7	12.2	79	86
14	1.8	4	3	75.0	11.3	63	60
15	1.9	4	4	100.0	9.9	62	54
16	1.6	5	5	100.0	10.9	62	51
17	1.6	4	3	75.0	7.2	37	41
18	2.8	7	7	100.0	10.5	59	87
19	2.2	6	5	83.3	10.0	58	67

LNG	= Utterance length in seconds
WORDS	= Number of words in utterance
*COR	= Number of words correctly recognized
%COR	= % of words correctly recognized
R-TIME	= Number of times real time needed for recognition ADD 8.4 for ACS and LPC calculations.
*STATES	= Average number of states checked at each segment
SEGS	= Number of segments produced (LNGx100 = number of 10ms. time samples)

Total words =	106
Correct words =	90 = 84.9%
Correct utterances =	6 = 31.6%
Average search time =	9.9 times real time
Including ACS and LPC calculation =	18.3 times real time
Average number of states searched =	56
Ratio of segments/10ms samples =	29.7%

Test Detail of Speaker RG, No Syntax, on Test Data recorded 5 Months after Training Data.

```
1   UTT="STORE NEGATE ALPHA INTO EPSILON"
    REC="STORE MIN BETA ALPHA INTO EPSILON"                 CORRECT=4/5
2   UTT="PUT ABSOLUTE BETA IN DELTA"
    REC="WHAT ABSOLUTE BETA IN MOD"                         CORRECT=3/5
3   UTT="ALPHA GETS FACT GAMMA"
    REC="ALPHA GETS FACT GAMMA"                             CORRECT=4/4
4   UTT="BETA BECOMES OCTAL ONE ZERO"
    REC="BETA BECOMES OCTAL ONE ZERO"                       CORRECT=5/5
5   UTT="GAMMA GETS DECIMAL TWO THREE"
    REC="GAMMA GETS OCTAL TWO THREE"                        CORRECT=4/5
6   UTT="DELTA BECOMES FOUR PLUS FIVE"
    REC="DELTA BECOMES FOUR PLUS MOD"                       CORRECT=4/5
7   UTT="EPSILON GETS SIX EIGHT MINUS DELTA"
    REC="EPSILON GETS SIX EIGHT NINE IS MOD"                CORRECT=4/6
8   UTT="WHAT IS SEVEN NINE TIMES EPSILON"
    REC="WHAT IS GAMMA NINE TIMES EPSILON"                  CORRECT=5/6
9   UTT="SHOW ONE ZERO TWO DIVIDE THREE FOUR"
    REC="SHOW ONE ZERO TWO MOD DIVIDE THREE FOUR"           CORRECT=6/7
10  UTT="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"
    REC="STORE FIVE SIX TIMES SEVEN BETA INTO ALPHA"        CORRECT=6/8
11  UTT="PUT NINE POWER TWO ONE IN BETA"
    REC="WHAT NINE POWER TWO ONE IN BETA"                   CORRECT=6/7
12  UTT="ALPHA BECOMES THREE ZERO MAX GAMMA"
    REC="ALPHA BECOMES THREE ZERO MAX GAMMA"                CORRECT=6/6
13  UTT="BETA GETS DECIMAL FOUR SIX MIN DELTA"
    REC="BETA GETS DECIMAL FOUR SIX MIN ALPHA"              CORRECT=6/7
14  UTT="DELTA GETS ABSOLUTE ALPHA"
    REC="MOD GETS ABSOLUTE ALPHA"                           CORRECT=3/4
15  UTT="EPSILON BECOMES FACT BETA"
    REC="EPSILON BECOMES FACT BETA"                         CORRECT=4/4
16  UTT="WHAT IS SEVEN PLUS EIGHT"
    REC="WHAT IS SEVEN PLUS EIGHT"                          CORRECT=5/5
17  UTT="SHOW NINE MINUS FIVE"
    REC="SHOW NINE MINUS MOD"                               CORRECT=3/4
18  UTT="STORE TWO ZERO TIMES THREE INTO GAMMA"
    REC="STORE TWO ZERO TIMES THREE INTO GAMMA"             CORRECT=7/7
19  UTT="PUT FOUR DIVIDE DELTA IN EPSILON"
    REC="WHAT FOUR DIVIDE DELTA IN EPSILON"                 CORRECT=5/6
```

Test Summary of Speaker BL, With Syntax, on Test Data recorded 5 Months after Training Data.

UTT	LNG	WORDS	#COR	%COR	R-TIME	#STATES	SEGS
1	2.1	5	4	80.0	3.7	14	63
2	1.8	5	5	100.0	3.5	15	54
3	1.6	4	4	100.0	4.5	26	46
4	2.0	5	5	100.0	4.4	24	59
5	1.8	5	5	100.0	5.2	25	58
6	1.9	5	4	80.0	4.2	18	64
7	2.3	6	6	100.0	4.2	23	68
8	2.1	6	6	100.0	5.5	37	65
9	2.6	7	6	85.7	6.0	45	72
10	2.7	8	7	87.5	5.7	37	90
11	2.0	7	7	100.0	4.9	25	61
12	2.3	6	6	100.0	4.8	22	77
13	2.5	7	7	100.0	4.5	23	77
14	1.9	4	4	100.0	4.7	25	62
15	1.7	4	4	100.0	4.3	20	59
16	1.9	4	4	100.0	4.1	19	59
17	1.6	5	5	100.0	4.3	25	42
18	1.6	4	4	100.0	3.9	20	40
19	2.7	7	7	100.0	5.0	26	86
20	2.3	6	6	100.0	4.4	24	71

LNG	= Utterance length in seconds
WORDS	= Number of words in utterance
#COR	= Number of words correctly recognized
%COR	= % of words correctly recognized
R-TIME	= Number of times real time needed for recognition ADD 8.4 for ACS and LPC calculations.
#STATES	= Average number of states checked at each segment
SEGS	= Number of segments produced (LNGx100 = number of 10ms. time samples)

Total words =	110
Correct words =	106 = 96.4%
Correct utterances =	16 = 80.0%
Average search time =	4.7 times real time
Including ACS and LPC calculation =	13.1 times real time
Average number of states searched =	24
Ratio of segments/10ms samples =	30.9%

Test Detail of Speaker BL, With Syntax, on Test Data recorded 5 Months after Training Data.

```
1   UTT="STORE NEGATE ALPHA INTO EPSILON"
    REC="STORE NEGATE DELTA INTO EPSILON"              CORRECT=4/5
2   UTT="PUT ABSOLUTE BETA IN DELTA"
    REC="PUT ABSOLUTE BETA IN DELTA"                   CORRECT=5/5
3   UTT="ALPHA GETS FACT GAMMA"
    REC="ALPHA GETS FACT GAMMA"                        CORRECT=4/4
4   UTT="BETA BECOMES OCTAL ONE ZERO"
    REC="BETA BECOMES OCTAL ONE ZERO"                  CORRECT=5/5
5   UTT="GAMMA GETS DECIMAL TWO THREE"
    REC="GAMMA GETS DECIMAL TWO THREE"                 CORRECT=5/5
6   UTT="DELTA BECOMES FOUR PLUS FIVE"
    REC="DELTA BECOMES FOUR PLUS NINE"                 CORRECT=4/5
7   UTT="EPSILON GETS SIX EIGHT MINUS DELTA"
    REC="EPSILON GETS SIX EIGHT MINUS DELTA"           CORRECT=6/6
8   UTT="WHAT IS SEVEN NINE TIMES EPSILON"
    REC="WHAT IS SEVEN NINE TIMES EPSILON"             CORRECT=6/6
9   UTT="SHOW ONE ZERO TWO DIVIDE THREE FOUR"
    REC="SHOW ONE ZERO TWO DIVIDE THREE ONE"           CORRECT=6/7
10  UTT="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"
    REC="STORE FIVE SIX NINE SEVEN EIGHT INTO ALPHA"   CORRECT=7/8
11  UTT="PUT NINE POWER TWO ONE IN BETA"
    REC="PUT NINE POWER TWO ONE IN BETA"               CORRECT=7/7
12  UTT="ALPHA BECOMES THREE ZERO MAX GAMMA"
    REC="ALPHA BECOMES THREE ZERO MAX GAMMA"           CORRECT=6/6
13  UTT="BETA GETS DECIMAL FOUR SIX MIN DELTA"
    REC="BETA GETS DECIMAL FOUR SIX MIN DELTA"         CORRECT=7/7
14  UTT="GAMMA BECOMES NEGATE EPSILON"
    REC="GAMMA BECOMES NEGATE EPSILON"                 CORRECT=4/4
15  UTT="DELTA GETS ABSOLUTE ALPHA"
    REC="DELTA GETS ABSOLUTE ALPHA"                    CORRECT=4/4
16  UTT="EPSILON BECOMES FACT BETA"
    REC="EPSILON BECOMES FACT BETA"                    CORRECT=4/4
17  UTT="WHAT IS SEVEN PLUS EIGHT"
    REC="WHAT IS SEVEN PLUS EIGHT"                     CORRECT=5/5
18  UTT="SHOW NINE MINUS FIVE"
    REC="SHOW NINE MINUS FIVE"                         CORRECT=4/4
19  UTT="STORE TWO ZERO TIMES THREE INTO GAMMA"
    REC="STORE TWO ZERO TIMES THREE INTO GAMMA"        CORRECT=7/7
20  UTT="PUT FOUR DIVIDE DELTA IN EPSILON"
    REC="PUT FOUR DIVIDE DELTA IN EPSILON"             CORRECT=6/6
```

Test Summary of Speaker KP, With Syntax, on Test Data recorded 5 Months after Training Data.

UTT	LNG	WORDS	#COR	%COR	R-TIME	#STATES	SEGS
1	2.0	5	5	100.0	4.1	23	54
2	1.8	5	5	100.0	6.1	52	57
3	1.5	4	4	100.0	3.6	24	39
4	2.0	5	5	100.0	4.6	37	55
5	1.7	5	3	60.0	4.6	32	49
6	1.9	5	3	60.0	3.6	26	43
7	2.3	6	4	66.7	7.6	71	71
8	2.0	6	6	100.0	5.5	52	54
9	2.6	7	5	71.4	5.2	45	63
10	2.8	8	8	100.0	4.3	28	77
11	2.6	7	7	100.0	3.5	23	55
12	2.3	6	6	100.0	4.2	26	63
13	2.6	7	6	85.7	6.5	66	67
14	2.1	4	4	100.0	3.8	21	53
15	1.9	4	4	100.0	4.8	38	52
16	1.9	4	4	100.0	4.4	35	46
17	1.6	5	3	60.0	7.1	79	42
18	1.6	4	2	50.0	4.1	25	35
19	2.7	7	6	85.7	6.1	51	70
20	2.2	6	6	100.0	5.6	45	59

LNG	= Utterance length in seconds
WORDS	= Number of words in utterance
#COR	= Number of words correctly recognized
%COR	= % of words correctly recognized
R-TIME	= Number of times real time needed for recognition ADD 8.4 for ACS and LPC calculations.
#STATES	= Average number of states checked at each segment
SEGS	= Number of segments produced (LNGx100 = number of 10ms. time samples)

Total words =	110
Correct words =	96 = 87.3%
Correct utterances =	12 = 60.0%
Average search time =	5.0 times real time
Including ACS and LPC calculation =	13.4 times real time
Average number of states searched =	39
Ratio of segments/10ms samples =	26.2%

Test Detail of Speaker KP, With Syntax,
on Test Data recorded 5 Months after Training Data.

1	UTT="STORE NEGATE ALPHA INTO EPSILON"	
	REC="STORE NEGATE ALPHA INTO EPSILON"	CORRECT=5/5
2	UTT="PUT ABSOLUTE BETA IN DELTA"	
	REC="PUT ABSOLUTE BETA IN DELTA"	CORRECT=5/5
3	UTT="ALPHA GETS FACT GAMMA"	
	REC="ALPHA GETS FACT GAMMA"	CORRECT=4/4
4	UTT="BETA BECOMES OCTAL ONE ZERO"	
	REC="BETA BECOMES OCTAL ONE ZERO"	CORRECT=5/5
5	UTT="GAMMA GETS DECIMAL TWO THREE"	
	REC="GAMMA BECOMES ONE TWO THREE"	CORRECT=3/5
6	UTT="DELTA BECOMES FOUR PLUS FIVE"	
	REC="DELTA BECOMES FOUR MOD NINE"	CORRECT=3/5
7	UTT="EPSILON GETS SIX EIGHT MINUS DELTA"	
	REC="EPSILON GETS EIGHT EIGHT NINE MAX DELTA"	CORRECT=4/6
8	UTT="WHAT IS SEVEN NINE TIMES EPSILON"	
	REC="WHAT IS SEVEN NINE TIMES EPSILON"	CORRECT=6/6
9	UTT="SHOW ONE ZERO TWO DIVIDE THREE FOUR"	
	REC="SHOW ONE ZERO TWO TWO NINE THREE ONE"	CORRECT=5/7
10	UTT="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"	
	REC="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"	CORRECT=8/8
11	UTT="PUT NINE POWER TWO ONE IN BETA"	
	REC="PUT NINE POWER TWO ONE IN BETA"	CORRECT=7/7
12	UTT="ALPHA BECOMES THREE ZERO MAX GAMMA"	
	REC="ALPHA BECOMES THREE ZERO MAX GAMMA"	CORRECT=6/6
13	UTT="BETA GETS DECIMAL FOUR SIX MIN DELTA"	
	REC="BETA GETS DECIMAL ONE SIX MIN DELTA"	CORRECT=6/7
14	UTT="GAMMA BECOMES NEGATE EPSILON"	
	REC="GAMMA BECOMES NEGATE EPSILON"	CORRECT=4/4
15	UTT="DELTA GETS ABSOLUTE ALPHA"	
	REC="DELTA GETS ABSOLUTE ALPHA"	CORRECT=4/4
16	UTT="EPSILON BECOMES FACT BETA"	
	REC="EPSILON BECOMES FACT BETA"	CORRECT=4/4
17	UTT="WHAT IS SEVEN PLUS EIGHT"	
	REC="WHAT IS ONE ONE MOD EIGHT"	CORRECT=3/5
18	UTT="SHOW NINE MINUS FIVE"	
	REC="SHOW NINE NINE MIN ONE"	CORRECT=2/4
19	UTT="STORE TWO ZERO TIMES THREE INTO GAMMA"	
	REC="STORE TWO ZERO TWO ONE THREE INTO GAMMA"	CORRECT=6/7
20	UTT="PUT FOUR DIVIDE DELTA IN EPSILON"	
	REC="PUT FOUR DIVIDE DELTA IN EPSILON"	CORRECT=6/6

Test Summary of Speaker DS, With Syntax,
on Test Data recorded 5 Months after Training Data.

UTT	LNG	WORDS	#COR	%COR	R-TIME	#STATES	SEGS
1	2.1	5	4	80.0	3.9	20	61
2	1.9	5	5	100.0	3.9	20	57
3	1.8	4	4	100.0	3.6	22	47
4	2.3	5	5	100.0	4.3	26	69
5	2.2	5	5	100.0	4.4	27	61
6	2.1	5	5	100.0	3.7	22	54
7	2.6	6	6	100.0	4.2	26	75
8	2.2	6	6	100.0	3.9	22	63
9	2.8	7	6	85.7	5.8	42	82
10	3.1	8	8	100.0	4.5	30	86
11	2.4	7	7	100.0	5.2	34	72
12	2.5	6	5	83.3	4.2	21	75
13	2.7	7	7	100.0	4.8	34	79
14	2.0	4	4	100.0	4.7	30	64
15	1.9	4	4	100.0	3.9	21	62
16	2.0	4	4	100.0	4.4	28	60
17	1.7	5	4	80.0	5.6	47	49
18	1.8	4	3	75.0	4.7	37	46
19	2.7	7	5	71.4	6.0	40	86
20	2.3	6	6	100.0	6.1	51	73

LNG	= Utterance length in seconds
WORDS	= Number of words in utterance
#COR	= Number of words correctly recognized
%COR	= % of words correctly recognized
R-TIME	= Number of times real time needed for recognition ADD 8.4 for ACS and LPC calculations.
#STATES	= Average number of states checked at each segment
SEGS	= Number of segments produced (LNGx100 = number of 10ms. time samples)

Total words =	110
Correct words =	103 = 93.6%
Correct utterances =	14 = 70.0%
Average search time =	4.6 times real time
Including ACS and LPC calculation =	13.0 times real time
Average number of states searched =	30
Ratio of segments/10ms samples =	29.2%

Test Detail of Speaker DS, With Syntax, on Test Data recorded 5 Months after Training Data.

```
1    UTT="STORE NEGATE ALPHA INTO EPSILON"
     REC="STORE NEGATE ALPHA IN EPSILON"                CORRECT=4/5
2    UTT="PUT ABSOLUTE BETA IN DELTA"
     REC="PUT ABSOLUTE BETA IN DELTA"                   CORRECT=5/5
3    UTT="ALPHA GETS FACT GAMMA"
     REC="ALPHA GETS FACT GAMMA"                        CORRECT=4/4
4    UTT="BETA BECOMES OCTAL ONE ZERO"
     REC="BETA BECOMES OCTAL ONE ZERO"                  CORRECT=5/5
5    UTT="GAMMA GETS DECIMAL TWO THREE"
     REC="GAMMA GETS DECIMAL TWO THREE"                 CORRECT=5/5
6    UTT="DELTA BECOMES FOUR PLUS FIVE"
     REC="DELTA BECOMES FOUR PLUS FIVE"                 CORRECT=5/5
7    UTT="EPSILON GETS SIX EIGHT MINUS DELTA"
     REC="EPSILON GETS SIX EIGHT MINUS DELTA"           CORRECT=6/6
8    UTT="WHAT IS SEVEN NINE TIMES EPSILON"
     REC="WHAT IS SEVEN NINE TIMES EPSILON"             CORRECT=6/6
9    UTT="SHOW ONE ZERO TWO DIVIDE THREE FOUR"
     REC="SHOW NINE ZERO TWO DIVIDE THREE FOUR"         CORRECT=6/7
10   UTT="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"
     REC="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"    CORRECT=8/8
11   UTT="PUT NINE POWER TWO ONE IN BETA"
     REC="PUT NINE POWER TWO ONE IN BETA"               CORRECT=7/7
12   UTT="ALPHA BECOMES THREE ZERO MAX GAMMA"
     REC="ALPHA BECOMES THREE ZERO MAX ZERO"            CORRECT=5/6
13   UTT="BETA GETS DECIMAL FOUR SIX MIN DELTA"
     REC="BETA GETS DECIMAL FOUR SIX MIN DELTA"         CORRECT=7/7
14   UTT="GAMMA BECOMES NEGATE EPSILON"
     REC="GAMMA BECOMES NEGATE EPSILON"                 CORRECT=4/4
15   UTT="DELTA GETS ABSOLUTE ALPHA"
     REC="DELTA GETS ABSOLUTE ALPHA"                    CORRECT=4/4
16   UTT="EPSILON BECOMES FACT BETA"
     REC="EPSILON BECOMES FACT BETA"                    CORRECT=4/4
17   UTT="WHAT IS SEVEN PLUS EIGHT"
     REC="WHAT IS SEVEN PLUS BETA"                      CORRECT=4/5
18   UTT="SHOW NINE MINUS FIVE"
     REC="SHOW NINE MINUS ALPHA"                        CORRECT=3/4
19   UTT="STORE TWO ZERO TIMES THREE INTO GAMMA"
     REC="STORE TWO ZERO TIMES FOUR EIGHT IN GAMMA"     CORRECT=5/7
20   UTT="PUT FOUR DIVIDE DELTA IN EPSILON"
     REC="PUT FOUR DIVIDE DELTA IN EPSILON"             CORRECT=6/6
```

Test Summary of Speaker RG, With Syntax, on Test Data recorded 5 Months after Training Data.

UTT	LNG	WORDS	#COR	%COR	R-TIME	#STATES	SEGS
1	1.9	5	5	100.0	3.6	16	56
2	1.7	5	5	100.0	4.0	19	54
3	1.5	4	4	100.0	3.5	23	38
4	1.9	5	5	100.0	4.3	22	57
5	1.9	5	4	80.0	5.4	38	58
6	1.7	5	4	80.0	4.2	23	53
7	2.4	6	6	100.0	5.3	37	76
8	2.0	6	6	100.0	4.8	41	52
9	2.7	7	6	85.7	4.6	26	77
10	3.1	8	7	87.5	5.7	42	91
11	2.1	7	7	100.0	4.2	22	65
12	2.6	6	6	100.0	4.2	26	68
13	2.7	7	6	85.7	6.2	47	86
14	1.8	4	4	100.0	5.4	34	60
15	1.9	4	4	100.0	4.2	25	54
16	1.6	5	5	100.0	5.1	30	51
17	1.6	4	4	100.0	3.6	18	41
18	2.8	7	7	100.0	5.2	32	87
19	2.2	6	6	100.0	4.7	30	67

LNG	= Utterance length in seconds
WORDS	= Number of words in utterance
#COR	= Number of words correctly recognized
%COR	= % of words correctly recognized
R-TIME	= Number of times real time needed for recognition ADD 8.4 for ACS and LPC calculations.
#STATES	= Average number of states checked at each segment
SEGS	= Number of segments produced (LNGx100 = number of 10ms. time samples)

Total words =	106
Correct words =	101 = 95.3%
Correct utterances =	14 = 73.7%
Average search time =	4.7 times real time
Including ACS and LPC calculation =	13.1 times real time
Average number of states searched =	29
Ratio of segments/10ms samples =	29.7%

Test Detail of Speaker RG, With Syntax, on Test Data recorded 5 Months after Training Data.

```
1   UTT="STORE NEGATE ALPHA INTO EPSILON"
    REC="STORE NEGATE ALPHA INTO EPSILON"                      CORRECT=5/5
2   UTT="PUT ABSOLUTE BETA IN DELTA"
    REC="PUT ABSOLUTE BETA IN DELTA"                           CORRECT=5/5
3   UTT="ALPHA GETS FACT GAMMA"
    REC="ALPHA GETS FACT GAMMA"                                CORRECT=4/4
4   UTT="BETA BECOMES OCTAL ONE ZERO"
    REC="BETA BECOMES OCTAL ONE ZERO"                          CORRECT=5/5
5   UTT="GAMMA GETS DECIMAL TWO THREE"
    REC="GAMMA GETS OCTAL TWO THREE"                           CORRECT=4/5
6   UTT="DELTA BECOMES FOUR PLUS FIVE"
    REC="DELTA BECOMES FOUR PLUS ONE"                          CORRECT=4/5
7   UTT="EPSILON GETS SIX EIGHT MINUS DELTA"
    REC="EPSILON GETS SIX EIGHT MINUS DELTA"                   CORRECT=6/6
8   UTT="WHAT IS SEVEN NINE TIMES EPSILON"
    REC="WHAT IS SEVEN NINE TIMES EPSILON"                     CORRECT=6/6
9   UTT="SHOW ONE ZERO TWO DIVIDE THREE FOUR"
    REC="SHOW ONE ZERO TWO MOD TWO NINE THREE FOUR"            CORRECT=6/7
10  UTT="STORE FIVE SIX MOD SEVEN EIGHT INTO ALPHA"
    REC="STORE FIVE SIX TIMES SEVEN EIGHT INTO ALPHA"          CORRECT=7/8
11  UTT="PUT NINE POWER TWO ONE IN BETA"
    REC="PUT NINE POWER TWO ONE IN BETA"                       CORRECT=7/7
12  UTT="ALPHA BECOMES THREE ZERO MAX GAMMA"
    REC="ALPHA BECOMES THREE ZERO MAX GAMMA"                   CORRECT=6/6
13  UTT="BETA GETS DECIMAL FOUR SIX MIN DELTA"
    REC="BETA GETS DECIMAL FOUR SIX MIN ALPHA"                 CORRECT=6/7
14  UTT="DELTA GETS ABSOLUTE ALPHA"
    REC="DELTA GETS ABSOLUTE ALPHA"                            CORRECT=4/4
15  UTT="EPSILON BECOMES FACT BETA"
    REC="EPSILON BECOMES FACT BETA"                            CORRECT=4/4
16  UTT="WHAT IS SEVEN PLUS EIGHT"
    REC="WHAT IS SEVEN PLUS EIGHT"                             CORRECT=5/5
17  UTT="SHOW NINE MINUS FIVE"
    REC="SHOW NINE MINUS FIVE"                                 CORRECT=4/4
18  UTT="STORE TWO ZERO TIMES THREE INTO GAMMA"
    REC="STORE TWO ZERO TIMES THREE INTO GAMMA"                CORRECT=7/7
19  UTT="PUT FOUR DIVIDE DELTA IN EPSILON"
    REC="PUT FOUR DIVIDE DELTA IN EPSILON"                     CORRECT=6/6
```

Network Statistics

TASK	STATES	POINTERS	PHONS	TEMPLATES
BAKER CHESS	410	481	33	168
BAKER DOCTOR	702	775	33	168
BAKER NEWS	498	558	33	168
BAKER FORMANT	2356	2256	33	168
BAKER DESK CALC	466	1065	33	168
QTRAIN CHESS	323	1020	62	83
QTRAIN DOCTOR	592	1147	62	83
QTRAIN NEWS	439	884	62	83
QTRAIN FORMANT	1964	6027	62	83
QTRAIN DESK CALC	973	5040	62	83
DESK CALC	820	3497	67	67
DESK CALC NO SYNTAX	244	2271	67	67

Large Grammars, Preliminary Results

The following are preliminary results for a task grammar of 1011 words and a branching factor of about 10. These results were obtained by a PDP KL-10 computer which is about four times faster than the KA-10 computer which was used to obtain the previous results. Therefore, the recognition times should be multiplied by 4 to obtain comparable recognition times. The network for this grammar contains about 18000 states and about 29000 pointers. This is the largest network ever attempted with the Harpy system. The results are for one typical data set of about 25 utterances.

Complete results and details for the large grammars will be given in future editions of the "Working Papers in Speech Recognition", published by the Computer Science Department, Carnegie-Mellon University.

Preliminary Test Summary of Large Grammar

UTT	LNG	WORDS	#COR	%COR	R-TIME	#STATES	SEGS
1	2.5	7	7	100.0	7.9	72	87
2	3.0	6	6	100.0	6.5	29	103
3	2.5	6	5	83.3	7.7	55	89
4	2.6	8	6	75.0	7.2	56	88
5	3.5	8	8	100.0	6.4	46	111
6	2.8	7	7	100.0	6.7	36	95
7	1.3	3	3	100.0	3.9	6	31
8	1.7	5	5	100.0	5.5	30	50
9	2.7	8	8	100.0	5.7	16	89
10	2.9	7	7	100.0	5.8	39	88
11	2.0	6	6	100.0	5.4	34	58
12	1.6	5	5	100,0	6.1	42	48
13	1.5	5	5	100.0	4,9	19	42
14	1.5	6	6	100.0	5.7	18	47
15	2.4	9	9	100.0	6.2	26	82
16	2.0	5	5	100.0	5.9	19	69
17	1.9	8	4	12.5	5.8	33	58
18	2.2	7	7	100.0	6.3	40	73
19	1.4	4	0	0.0	7.5	112	38
20	1.5	6	6	100.0	7.0	74	47
21	0.7	1	1	100.0	4.2	17	15
22	1.2	4	4	100.0	5.6	38	34
23	1.1	3	3	100.0	5.9	29	33
24	1.1	4	4	100.0	7.8	88	34
25	2.6	8	8	100.0	5.9	46	77

LNG	= Utterance length in seconds
WORDS	= Number of words in utterance
#COR	= Number of words correctly recognized
%COR	= % of words correctly recognized
R-TIME	= Number of times real time needed for recognition ADD 2.1 for ACS and LPC calculations.
#STATES	= Average number of states checked at each segment
SEGS	= Number of segments produced (LNGx100 = number of 10ms. time samples)

Total words =	146
Correct words =	132 = 90.4%
Correct utterances =	21 = 84.0%
Average search time =	6.3 times real time
Including ACS and LPC calculation =	8.4
Average number of states searched =	40
Ratio of segments/10ms samples =	32.0%

Preliminary Test Detail of Large Grammar

```
1   UTT="WHAT PAPERS ON GRAMMATICAL INFERENCE ARE THERE"
    REC="WHAT PAPERS ON GRAMMATICAL INFERENCE ARE THERE"    CORRECT=7/7
2   UTT="ANY ABSTRACTS REFERRING TO DYNAMIC CLUSTERING"
    REC="ANY ABSTRACTS REFERRING TO DYNAMIC CLUSTERING"     CORRECT=6/6
3   UTT="WHICH PAPERS CITE FEIGENBAUM AND FELDMAN"
    REC="WHICH PAPERS BY FEIGENBAUM AND FERDMAN"            CORRECT=5/6
4   UTT="ARE THERE ANY NEW PAPERS ON GRAPH MATCHING"
    REC="ARE THERE ANY NEW PAPERS ON DRAGON MENTIONED"      CORRECT=6/8
5   UTT="IS RESOLUTION THEOREM PROVING MENTIONED IN AN ABSTRACT"
    REC="IS RESOLUTION THEOREM PROVING MENTIONED IN AN ABSTRACT"
                                                            CORRECT=8/8
6   UTT="GET ME EVERYTHING ON UNIFORM PROOF PROCEDURES"
    REC="GET ME EVERYTHING ON UNIFORM PROOF PROCEDURES"     CORRECT=7/7
7   UTT="NO MORE PLEASE"
    REC="NO MORE PLEASE"                                    CORRECT=3/3
8   UTT="GIVE ME ONE MORE PLEASE"
    REC="GIVE ME ONE MORE PLEASE"                           CORRECT=5/5
9   UTT="WHO WROTE PAPERS ON PRODUCTION SYSTEMS THIS YEAR"
    REC="WHO WROTE PAPERS ON PRODUCTION SYSTEMS THIS YEAR"  CORRECT=8/8
10  UTT="DID ANY AI JOURNAL PAPERS CITE WOODS"
    REC="DID ANY AI JOURNAL PAPERS CITE WOODS"              CORRECT=7/7
11  UTT="DO ALL QUERIES TAKE THIS LONG"
    REC="DO ALL QUERIES TAKE THIS LONG"                     CORRECT=6/6
12  UTT="ARE YOU ALWAYS THIS SLOW"
    REC="ARE YOU ALWAYS THIS SLOW"                          CORRECT=5/5
13  UTT="HOW LONG DOES IT TAKE"
    REC="HOW LONG DOES IT TAKE"                             CORRECT=5/5
14  UTT="WHEN WILL YOU HAVE THE ANSWER"
    REC="WHEN WILL YOU HAVE THE ANSWER"                     CORRECT=6/6
15  UTT="DOES IT ALWAYS TAKE THIS LONG TO ANSWER ME"
    REC="DOES IT ALWAYS TAKE THIS LONG TO ANSWER ME"        CORRECT=9/9
16  UTT="DO RESPONSES EVER COME FASTER"
    REC="DO RESPONSES EVER COME FASTER"                     CORRECT=5/5
17  UTT="WHAT CAN I DO TO SPEED YOU UP"
    REC="WHAT HAVE THIS YEAR"                               CORRECT=1/8
18  UTT="HOW CAN I USE THE SYSTEM EFFICIENTLY"
    REC="HOW CAN I USE THE SYSTEM EFFICIENTLY"              CORRECT=7/7
19  UTT="WHAT MUST I ASK"
    REC="WHEN WAS AI AND"                                   CORRECT=0/4
20  UTT="WHAT DO I HAVE TO DO"
    REC="WHAT DO I HAVE TO DO"                              CORRECT=6/6
21  UTT="HELP"
    REC="HELP"                                              CORRECT=1/1
22  UTT="CAN YOU HELP ME"
    REC="CAN YOU HELP ME"                                   CORRECT=4/4
23  UTT="PLEASE HELP ME"
```

```
        REC="PLEASE HELP ME"                                      CORRECT=3/3
14      UTT="WHAT SHOULD I ASK"
        REC="WHAT SHOULD I ASK"                                   CORRECT=4/4
15      UTT="WHEN WAS THE LAST PAPER BY HOLLAND PUBLISHED"
        REC="WHEN WAS THE LAST PAPER BY HOLLAND PUBLISHED"        CORRECT=8/8
```